



**INTEGRATION OF THE NETWORK AND  
APPLICATION LAYERS OF  
AUTOMATICALLY-  
CONFIGURED PROGRAMMABLE LOGIC  
CONTROLLER HONEYPOTS**

THESIS

Justin K. Gallenstein, 2d Lt, USAF  
AFIT-ENG-MS-17-M-029

**DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY**

***AIR FORCE INSTITUTE OF TECHNOLOGY***

**Wright-Patterson Air Force Base, Ohio**

DISTRIBUTION STATEMENT A  
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-17-M-029

INTEGRATION OF THE NETWORK AND APPLICATION LAYERS OF  
AUTOMATICALLY-CONFIGURED PROGRAMMABLE LOGIC CONTROLLER  
HONEYPOTS

THESIS

Presented to the Faculty  
Department of Electrical and Computer Engineering  
Graduate School of Engineering and Management  
Air Force Institute of Technology  
Air University  
Air Education and Training Command  
in Partial Fulfillment of the Requirements for the  
Degree of Master of Science in Cyber Operations

Justin K. Gallenstein, 2d Lt, USAF, B.S.C.S

March 2017

DISTRIBUTION STATEMENT A  
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-MS-17-M-029

INTEGRATION OF THE NETWORK AND APPLICATION LAYERS OF  
AUTOMATICALLY-CONFIGURED PROGRAMMABLE LOGIC CONTROLLER  
HONEYPOTS

THESIS

Justin K. Gallenstein, 2d Lt, USAF, B.S.C.S

Committee Membership:

Barry E. Mullins, Ph.D., P.E.

(Chairman)

LTC Mason J. Rice, Ph.D.

(Member)

Juan Lopez Jr., Ph.D.

(Member)

## Abstract

Much of the critical infrastructure of the world is controlled by programmable logic controllers (PLC). These PLCs regulate the processes of these industries, and therefore are targets for malicious actors around the globe. Honeypots are one of various security mechanisms that can be deployed to help protect these vital systems. In order to work, a honeypot must accurately mimic the system under protection. However, within the PLC market there are numerous manufacturers and protocols which makes mimicking PLCs using one monolithic software package a daunting task. To mitigate this shortfall, ScriptGenE, a protocol-agnostic framework capable of accurately creating PLC honeypots, is designed.

ScriptGenE uses previously captured PLC traffic to create a tree of the protocol and selectively respond to application layer requests in an accurate way. This research integrates ScriptGenE with Honeyd to provide the PLC honeypots with an accurate network layer. This combination provides a comprehensive PLC honeypot.

Testing is done by using the combined framework to emulate a network of Allen-Bradley ControlLogix, Allen-Bradley CompactLogix, and Siemens S7-300 PLCs. A series of tools are used to evaluate the legitimacy of the emulated PLC network including Nmap, Honeyscore, RSLinx, STEP7, and Wget. Nmap and Honeyscore are used to show that the combined framework is able to accurately emulate the network layer of three different PLC types with 100 percent accuracy. Using Wget, RSLinx, and STEP7, this research shows the ability to emulate more advanced application layer protocols such as ENIP, ISOTASP, and HTTP with accuracies of 78, 100, and 67 percent respectively. This completed framework provides a viable solution to help protect critical infrastructure around the world.

AFIT-ENG-MS-17-M-029

*To my family, for their boundless support.*

## Acknowledgements

I would like to thank Phillip Warner and Kyle Girtz for their previous work on this project that laid the foundation for my efforts. I would also thank Dr. Mullins for simultaneously providing me support and autonomy. Lastly, I would like to thank the other members in my committee, LTC Mason Rice and Mr. Juan Lopez, for sharing their expertise and time.

Justin K. Gallenstein, 2d Lt, USAF

# Table of Contents

	Page
Abstract .....	iv
Dedication .....	v
Acknowledgements .....	vi
List of Figures .....	x
List of Tables .....	xii
List of Acronyms .....	xiii
I. Introduction .....	1
1.1 Background .....	1
1.2 Motivation .....	2
1.3 Research Goals and Hypothesis .....	3
1.4 Approach .....	4
1.4.1 ScriptGenE Enhancements .....	4
1.4.2 Honeyd Integration .....	5
1.4.3 Experimentation .....	5
1.5 Assumptions and Limitations .....	7
1.5.1 Network Protocols Involved .....	7
1.5.2 Limited Set of Tasks .....	7
1.5.3 Limited Set of Scanning Tools .....	8
1.5.4 Limited Configuration Setup .....	8
1.5.5 Resource Management .....	8
1.6 Thesis Overview .....	8
II. Background and Related Research .....	9
2.1 Overview .....	9
2.2 Background .....	9
2.2.1 Industrial Control Systems .....	9
2.2.2 ICS Vulnerabilities .....	12
2.2.3 Protocol Stack .....	15
2.2.4 Honeyd .....	16
2.2.5 Evaluation Tools .....	18
2.3 Related Research .....	21
2.3.1 Honeyd .....	21
2.3.2 Automatic Protocol Emulation .....	23
2.3.3 Advanced Replay Honeyd .....	24
2.3.4 ICS Honeyd .....	26



	Page
2.4 Chapter Summary .....	28
III. Framework Integration .....	30
3.1 Overview .....	30
3.2 Motivation and Application .....	30
3.3 The ScriptGenE Framework .....	30
3.3.1 Framework Overview .....	30
3.3.2 ScriptGenE.py .....	31
3.3.3 ScriptGenEemulate.py .....	35
3.4 ScriptGenE Framework Changes .....	35
3.4.1 UDP Support .....	37
3.4.2 Improved Backtracking Algorithm .....	37
3.4.3 Automatic Creation of Honeyd Profile .....	41
3.4.4 ScriptGenEemulate.py Alterations .....	41
3.4.5 Honeyd Integration .....	43
3.5 Design Summary .....	44
IV. Research Methodology .....	46
4.1 Goals .....	46
4.2 Approach .....	46
4.3 System Boundaries .....	50
4.4 Parameters and Factors .....	51
4.4.1 Workload Parameters .....	51
4.4.2 System Parameters .....	54
4.5 Performance Metrics .....	56
4.6 Experimental Setup .....	58
4.6.1 Overview .....	58
4.6.2 Machine Configurations .....	58
4.6.3 Experimental Scripts .....	60
4.6.4 GUI Automation .....	62
4.7 Methodology Summary .....	62
V. Results and Analysis .....	63
5.1 Overview .....	63
5.2 Metric 1 - Nmap .....	63
5.3 Metric 2 - Honeyscore .....	65
5.4 Metric 3 - PLC Module Discovery .....	67
5.4.1 RSLinx Tasks .....	69
5.4.2 STEP7 Tasks .....	71
5.5 Metric 4 - Best-Fit Algorithm .....	72

	Page
VI. Conclusions .....	78
6.1 Introduction .....	78
6.2 Research Conclusions .....	78
6.2.1 Integrated Emulation Performance .....	78
6.2.2 Best-Fit Algorithm .....	78
6.3 Significance of Research .....	79
6.3.1 Contributions .....	79
6.3.2 Applications .....	79
6.4 Future Work .....	80
6.4.1 Testing .....	80
6.4.2 Enhancing ScriptGenEmulate.py Algorithms .....	80
6.4.3 Honeyd Connection Overload .....	81
6.4.4 Manual P-tree Editing .....	81
6.5 Chapter Summary .....	82
Bibliography .....	83

## List of Figures

Figure		Page
1	Non-Integrated Nmap Results .....	6
2	ICS Block Diagram .....	11
3	General SCADA Layout .....	12
4	NIST Recommended ICS Network Configuration .....	13
5	Internet Protocol Stack .....	16
6	Nmap Results .....	19
7	Allen-Bradley Nmap Scan .....	19
8	Honeyscore Interface .....	20
9	Honeyd Configuration .....	21
10	Honeyd Architecture .....	22
11	Example Honeyd+ Production Configuration .....	27
12	ScriptGenE Framework Overview .....	31
13	Protocol Tree Structure .....	32
14	Example Segment of ENIP Protocol Tree .....	33
15	Example Segment of HTTP Protocol Tree .....	34
16	ScriptGenEemulate.py Usage .....	36
17	Heatmap Example .....	39
18	Matching Algorithm Example .....	40
19	Removed ScriptGenEemulate.py Socket Options .....	42
20	Non-Protocol Agnostic Code Segment .....	43
21	Fixed Honeyd Code Example .....	44
22	Zero Window Error .....	44

Figure		Page
23	Training Prison Network .....	47
24	Training Prison Network HMI .....	48
25	Honeyd Configuration File .....	49
26	ScriptGenE Emulator Framework .....	50
27	Nmap NSE enip-info Results .....	52
28	Nmap NSE s7-info Results .....	52
29	Possible Honeyscore Results .....	56
30	Successful and Failed Module Hierarchy Browsing in RSLinx .....	57
31	Successful and Failed Module Hierarchy Browsing in STEP7 .....	57
32	Experiment Setup .....	58
33	Altered Honeyscore Experiment Setup .....	59
34	Sikulix Code Snippet .....	62
35	Basic Nmap Scan Comparisons .....	64
36	Nmap NSE Script Comparisons .....	66
37	Honeyscore Results .....	67
38	Module Display Task Comparison .....	68
39	Attribute All Request Sequence .....	70
40	Send RR Data Request Sequence .....	70
41	Series of Resets by Legitimate PLC .....	70
42	RSLinx Module Display Graph .....	71
43	Web Server Result Comparison .....	75
44	Honeyd Data Acknowledgment .....	76
45	Protocol Tree Comparison .....	77

## List of Tables

Table		Page
1	System Under Test VM Configuration .....	59
2	RSLogix Windows XP VM Configuration .....	60
3	STEP7 Windows XP VM Configuration .....	60
4	Tool Hosting Linux VM Configuration .....	60
5	Nmap Experiment Results .....	63
6	NSE enip-info Results .....	65
7	NSE s7-info Results .....	65
8	RSLinux Results .....	69
9	STEP7 Results .....	72
10	Wget Results .....	73

## List of Acronyms

<b>AFIT</b>	Air Force Institute of Technology .....	2
<b>ARP</b>	Address Resolution Protocol.....	22
<b>CUT</b>	Component Under Test .....	50
<b>DCS</b>	Distributed Control System.....	10
<b>ENIP</b>	EtherNet Industrial Protocol.....	7
<b>FTP</b>	File Transfer Protocol .....	22
<b>HI</b>	High-Interaction .....	1
<b>HMI</b>	Human-Machine Interface .....	10
<b>HTML</b>	Hypertext Markup Language .....	53
<b>HTTP</b>	Hypertext Transfer Protocol .....	7
<b>ICS</b>	Industrial Control Systems.....	1
<b>ICMP</b>	Internet Control Message Protocol.....	22
<b>IDS</b>	Intrusion Detection System .....	28
<b>IP</b>	Internet Protocol .....	10
<b>ISO-TSAP</b>	ISO Transport Service Access Point.....	7
<b>IT</b>	Information Technology.....	9
<b>LI</b>	Low-Interaction.....	1
<b>MAC</b>	Media Access Control.....	5
<b>NAT</b>	Network Address Translation .....	53
<b>NIST</b>	National Institute of Standards and Technology.....	10
<b>Nmap</b>	Network Mapper.....	5

<b>NSE</b>	Nmap Scripting Engine.....	51
<b>OS</b>	Operating System .....	5
<b>PCAP</b>	Packet Capture.....	30
<b>PI</b>	Protocol Informatics.....	32
<b>PLC</b>	Programmable Logic Controller .....	1
<b>QEMU</b>	Quick Emulator .....	17
<b>SCADA</b>	Supervisory Control and Data Acquisition .....	10
<b>SMTP</b>	Simple Mail Transfer Protocol .....	22
<b>SNMP</b>	Simple Network Management Protocol.....	26
<b>SUT</b>	System Under Test .....	50
<b>TCP</b>	Transmission Control Protocol.....	2
<b>UDP</b>	User Datagram Protocol .....	4
<b>VM</b>	Virtual Machine .....	5

# INTEGRATION OF THE NETWORK AND APPLICATION LAYERS OF AUTOMATICALLY-CONFIGURED PROGRAMMABLE LOGIC CONTROLLER HONEYPOTS

## I. Introduction

### 1.1 Background

Today's modern society relies on the reliable production and transportation of energy, consumer products, water, and communication. These critical industries depend on Industrial Control Systems (ICS) to automatically regulate the required production processes. At inception, these ICS were designed with availability as the primary driver without much concern for security [1]. However, as these ICS were integrated into traditional IT networks, security became a major concern. The ICS device most important to the security of an ICS network is the Programmable Logic Controller (PLC). PLCs are responsible for information collection and response actuation. Stuxnet, malware targeting PLCs, demonstrated that substantial destruction could result from PLC compromise [2, 3]. These ICS networks also operate in an environment where traditional security techniques such as updating and patching are not practical as they potentially could interfere with availability.

One technology that has been explored to mitigate this security pitfall is honeypots. Honeypots are systems designed to mimic real systems in an effort to lure attackers away from production systems [4]. Honeypots exist in many different variations but can most simply be divided into two categories, High-Interaction (HI) and Low-Interaction (LI) [5]. HI honeypots closely emulate the full capabilities of the



target device, while LI honeypots only emulate specific services. Depending on the category, each of these systems provides different benefits and pitfalls. Regardless, the quality of a honeypot is determined by its authenticity and flexibility [6, 7].

## 1.2 Motivation

A myriad of problems are encountered when applying honeypot technology to ICS networks. Paramount among these is PLCs are expensive which often prevents utilizing a true PLC as a HI honeypot device. Instead, LI techniques often must be utilized to select a subset of services to imitate. Honeyd is a framework that can be utilized for this purpose. Honeyd automatically emulates the network layer of the traditional OSI stack and provides application layer emulation through optional custom extensions [8]. Unfortunately the burden of creating these extensions falls directly on the administrator. This becomes even more difficult when considering that most ICS devices utilize proprietary protocols.

It has been shown that automated protocol reverse-engineering can assist in emulating unknown protocols [9]. This idea was utilized in the ScriptGen framework that automatically created Honeyd configurations [10]. The Air Force Institute of Technology (AFIT) project ScriptGenE further extended this idea by emulating the application protocols of PLCs by replaying previous Transmission Control Protocol (TCP) conversations [11].

Another technique involves proxying traffic from multiple emulated hosts to one real device. These hybrid honeypots combine the previous HI/LI techniques to provide the end user with the benefits of both, while minimizing the pitfalls. One example of this is Honeyd+ which utilizes Honeyd for network layer traffic, while sending application layer traffic to a real PLC [12]. This system relies on a series of search and replace functions to exchange environmental variables (e.g., MAC and IP address)

to match the desired honeypot. This technique provides a larger target surface for a smaller investment in hardware, as one legitimate PLC can be used as the back-end server to multiple emulated PLCs. Unfortunately Honeyd+ can overload the hardware with proxied traffic, and this technique still requires the expensive PLC back-end.

Advanced examples of hybrid honeypots include SGNET and GQ, which are complex honeypot systems, designed to lure attackers into executing malware [13, 14]. These projects are not open source, so this research is unable to assess their applicability to this research’s goals.

These examples demonstrate that there is a need for an open source hybrid honeypot. This research further develops ScriptGenE to improve suitability as an ICS honeypot framework.

### **1.3 Research Goals and Hypothesis**

This research aims to achieve the following goals:

1. Enhance ScriptGenE capabilities to allow for accurate best-fit responses when exact matches are not available.
2. Integrate ScriptGenE with Honeyd to enable emulation of the industrial network of a simulated prison.

Currently ScriptGenE only emulates the application layer, severely limiting the authenticity of the framework [8, 11]. By integrating ScriptGenE with Honeyd, each honeypot is assigned its own accurate network personality. Furthermore, the back-tracking algorithm is updated to provide a best-match functionality to replace the previously utilized default error message. The hypothesis is that with these modifi-

cations the framework is able to successfully emulate an industrial network of various PLC types.

## **1.4 Approach**

Based on the above goals, this research is divided into two main threads: ScriptGenE framework enhancements and Honeyd integration.

### **1.4.1 ScriptGenE Enhancements.**

The ScriptGenE framework is an iterative product of various researchers, each targeting different areas of improvement. This research focuses on three main improvements.

#### **1.4.1.1 Protocol-Agnostic Adaptability.**

This framework is intended to operate without information on the specific protocol being emulated. However, prior to enhancements the framework contained various code sections that required hard-coded information for correct protocol emulation. For example to emulate the ENIP protocol, one section required changing a variable from -1 to 1. This signaled to ScriptGenE to execute a certain logic path. These are removed in favor of protocol-agnostic solutions.

#### **1.4.1.2 UDP Protocol Support.**

ScriptGenE previously only supported TCP traffic emulation. This presented a pitfall in PLC emulation because the User Datagram Protocol (UDP) protocol is utilized for many higher-level ICS protocols. This research adds functionality to allow for the emulation of UDP protocols.

#### **1.4.1.3 Improved Unknown Emulation.**

ScriptGenE operates by matching requests from the client to messages it has stored in its protocol tree (p-tree). Previous iterations of the framework only concerned themselves with exact matches. If an exact match was not found, the honeypot returned an error to the client. This research expands this functionality by creating a matching algorithm to return the best match.

#### **1.4.2 Honeyd Integration.**

ScriptGenE was designed to be eventually integrated into a honeynet framework. Previously the framework only emulated at the application layer. Thus, any network layer scans would return the results of the host machine. For instance, this research utilized a Linux-based Virtual Machine (VM) for hosting the framework. Any Network Mapper (Nmap) scan on the IP address of the honeypot would return results similar to those shown in Figure 1. As can be seen, an attacker could clearly ascertain, from both the Media Access Control (MAC) address and Operating System (OS), that this is not a legitimate PLC but in fact a VM-based Linux machine. With this new integration the scan should return results consistent with the desired PLC. Also, Honeyd allows for simulating a variety of PLCs within a single VM, thus increasing its security footprint, as only one host is needed to simulate an entire network.

#### **1.4.3 Experimentation.**

Experimentation is divided into two main categories, protocol emulation and Honeyd integration. To evaluate these two categories, four experiments are carried out using a network of three legitimate PLCs —an Allen Bradley ControlLogix, Allen

```

Nmap scan report for 192.168.159.128
Host is up (0.00s latency).
PORT      STATE SERVICE
80/tcp    open  http
44818/tcp open  EtherNetIP-2
MAC Address: 00:0C:29:4F:3A:A8 (VMware)
Warning: OSScan results may be unreliable because we could not
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.6

```

Figure 1. Nmap scan of non-integrated ScriptGen

Bradley CompactLogix, and a Siemens S7-300 with discrete and analog input/output modules.

1. The first focuses on using Nmap to evaluate the authenticity of the Honeyd personality. This test involves scanning the legitimate PLCs to generate a emulation target. Then three Honeyd honeypots to emulate these PLCs are created and subsequently scanned. Using the respective pairs of scans, the similarity between the legitimate and Honeyd instances is determined.

2. To further evaluate the Honeyd integration, these previously created honeypots are subjected to Shodan's Honeyscore in the second experiment. This tool was developed to look for common honeypot characteristics and assign a score from 0-1 based on the likelihood of the target being a honeypot.

3. The next experiment is to ensure that integration with Honeyd does not interfere with application protocol emulation. The created honeypots are scanned with their respective industry standard tool, RSLinx or STEP7, and evaluated to determine if they display modules correctly.

4. The last experiment focuses on the accuracy of the improved backtracking algorithm. The improvement is determined by comparing the current research with the previous ScriptGenE incarnation. These tests are carried out by utilizing Wget to query each honeypot with GET requests, each containing a different User-Agent.

Based on the responses, the ability of the new framework to generate a best-match is evaluated.

## **1.5 Assumptions and Limitations**

This research attempts to minimize the assumptions and limitations that apply. However, due to the scope of this field of research, various assumptions and limitations are required.

### **1.5.1 Network Protocols Involved.**

Only the protocols relevant to the designed experiments are utilized in this research including the Hypertext Transfer Protocol (HTTP), ISO Transport Service Access Point (ISO-TSAP), and EtherNet Industrial Protocol (ENIP). The design of ScriptGenE ideally renders the protocol irrelevant to emulation as it performs its actions without regard to the specifics of the protocol. However, protocol replay failures are possible due to improper protocol interpretation.

### **1.5.2 Limited Set of Tasks.**

PLCs are complex systems with many different functional roles. The emulated PLCs are subjected to various tests to simulate an adversary's early reconnaissance actions. These tests are used to evaluate the honeypot's ability to imitate a legitimate PLC. Despite attempts to diversify, these tests are just a subset of a variety of tasks that could have been chosen.

### **1.5.3 Limited Set of Scanning Tools.**

This research utilizes a set of scanning tools (i.e., Nmap and Honeyscore) to evaluate the emulated PLC's network layer. This set of tools is a selection of popular options. There is no guarantee that the emulated PLC would respond appropriately to other tools.

### **1.5.4 Limited Configuration Setup.**

The chosen PLCs are manufactured by Siemens and Allen-Bradley. Despite being two of the most widely deployed brands, these are only two vendors from variety of other PLC manufacturers [15].

### **1.5.5 Resource Management.**

These experiments are carried out with the assumption of one client interacting at a time. These frameworks are both designed to incorporate threading and resource management to allow for multiple interacting clients. Pilot testing also showed no resulting degradation despite an increase in clients. However, without further analysis, this research's chosen experiments cannot guarantee this result.

## **1.6 Thesis Overview**

Chapter II contains an overview of ICS technology and related work on honeypots and their applications. Chapter III provides a description of the developed emulator configuration. Chapter IV details the experimental design with results in Chapter V. Chapter VI presents research conclusions and suggestions for future work.

## II. Background and Related Research

### 2.1 Overview

This chapter outlines the security concerns related to ICS equipment responsible for controlling critical infrastructure. These systems have adopted traditional Information Technology (IT) to increase efficiency and interconnectivity. The adoption of this technology, without proper implementation, leads to vulnerabilities in these critical systems. Honeypots have been used to counter this threat, and there are many different configurations that each provide via a varied set of customizable specifications. This customization is often time consuming but necessary to provide a credible honeypot. This has created an area of research in automatically configuring these honeypots regardless of the required protocol.

### 2.2 Background

#### 2.2.1 Industrial Control Systems.

The United States defines critical infrastructure as systems and assets, whether physical or virtual, so vital to the United States that the incapacity or destruction of such systems and assets would have a debilitating impact on security, national economic security, national public health or safety, or any combination of those matters [16]. These systems are further broken down by Presidential Policy Directive PPD-21, which defines 16 different critical infrastructure sectors including energy, water and wastewater, chemical, transportation, and food and agriculture [17]. These sectors have slowly adopted industrial control technology in order to achieve increased automation, efficiency, and maintainability [1].

There are many distinctions within the critical infrastructure sectors between the different computational devices used to control their equipment. These distinctions



are dictated by the requirements of their systems. In this thesis the term ICS, as defined by National Institute of Standards and Technology (NIST), is used to refer to all of these distinctions including Supervisory Control and Data Acquisition (SCADA), Distributed Control System (DCS), and PLC.

Initially these ICS systems were vendor specific resulting in many different protocols. However in order to realize further benefits (e.g., interconnectivity and cost reduction), these industries are moving towards lower cost IT solutions, relying on widely deployed solutions such as TCP/Internet Protocol (IP). Despite some positive implications, the transformation has brought about new vulnerabilities, including targeted attacks by malicious actors. Previously these systems existed isolated from the external networks using unconnected proprietary technology. However as a byproduct of IP, if they are incorrectly configured, these systems can be accessed remotely by malicious actors around the globe. Since these ICS devices are responsible for controlling much of the critical infrastructure of the US, these vulnerabilities, if exploited, could have serious repercussions.

As shown in Figure 2, ICS systems are composed of three main components: Human-Machine Interface (HMI), control loop, and remote diagnostics. The HMI is typically a console where the human can monitor and interact with the system components, in order to ensure correct operation. Within the control loop the controller continuously polls the sensors for data. This data is used by the controller to actuate the components of the process to achieve the desired outputs. This could be as simple as a controller monitoring a temperature gauge and activating a fan when within certain temperature parameters.

In a typical environment the diagram is actually more complex, as the ICS control loop is often located at remote sites as shown in Figure 3. This is because when looking at critical infrastructure, the measurements must be taken at various geo-

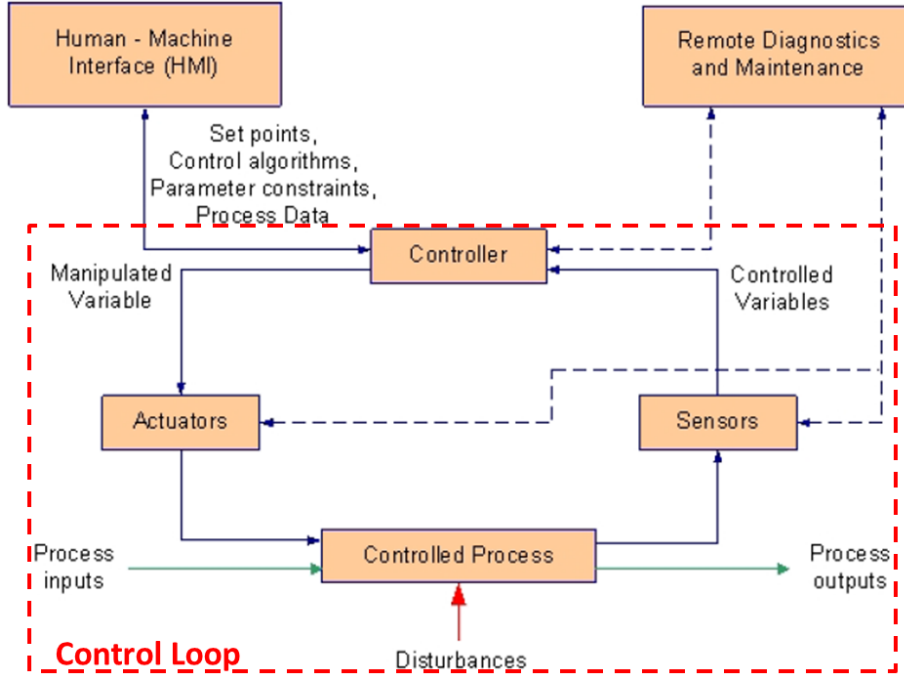


Figure 2. ICS block diagram [1]

graphic locations. Instead of having an individual HMI at each of these control loop locations, it is more efficient to have the data transmitted via Wide Area Networks (WAN) to a centralized HMI. This transmission over WAN takes place using various communication technologies. The use of these technologies can leave both the control center and remote sites vulnerable if not correctly implemented.

The picture gets even more complex when considering that these SCADA environments are often part of a much larger organizational structure. As Figure 4 shows, this structure often includes the corporate network, which for business purposes is exposed to the external Internet. This exposure means that potentially malicious actors could gain access to the sensitive control network. It is for this reason that NIST recommends using properly configured firewalls to divide the network appropriately.

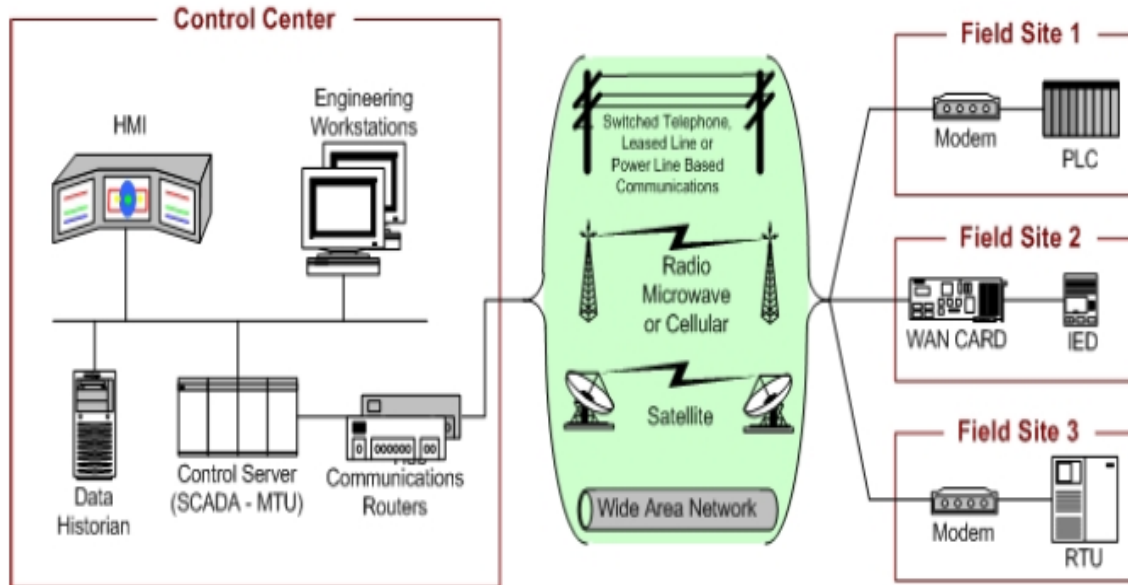


Figure 3. General SCADA layout [1]

### 2.2.2 ICS Vulnerabilities.

Finding these incorrectly-configured ICS devices has become even easier with the creation of Shodan, a search engine for embedded devices [18]. This engine can be used to enumerate a wide host of embedded devices including webcams, baby monitors, and most relevantly ICS equipment. Using this tool, adversaries can locate incorrectly-configured ICS equipment. Researchers have demonstrated this by using Shodan to create visualizations of over one million ICS devices [19, 20]. After finding these Internet-facing ICS devices, it has been shown that the function and sector of industry can be deduced which means attackers can target specific equipment based on motivation.[21]

A researcher at Trend Micro investigated the threat that Shodan posed to ICS devices by performing two studies. In these studies, Kyle Willhoit set up a combination of honeypots and actual hardware with insecure configurations. By monitoring these devices Willhoit determined there were 12 and 74 targeted attacks between the experiments, running one month and four months respectively [22, 23]. However, this

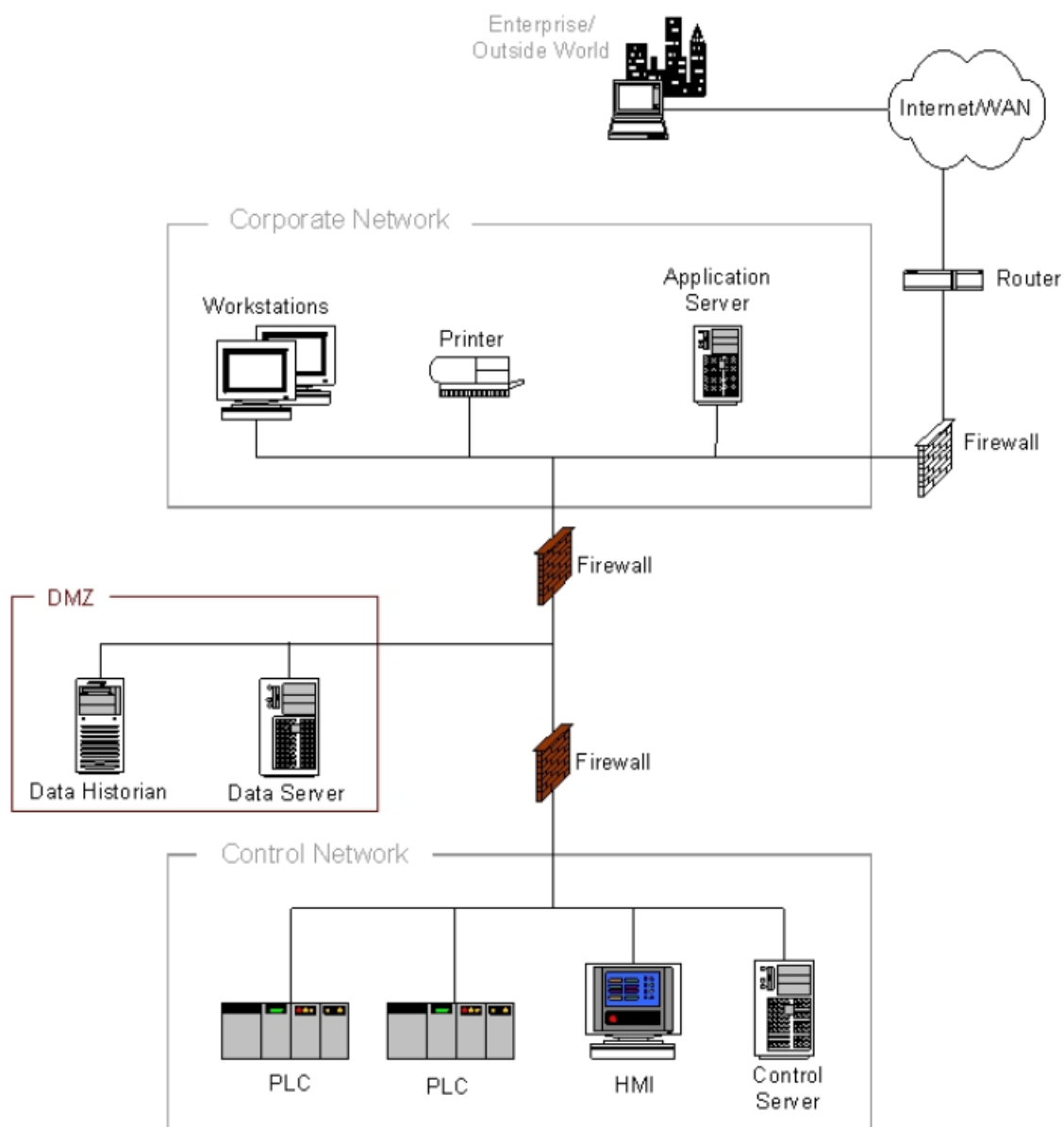


Figure 4. ICS network configuration recommended by NIST [1]

data was later contradicted by a experiment done by Bodenheimer, in which four different Internet-facing PLCs, with varying exposure levels, were monitored for attacks. After 55 days of monitoring there were no targeted attacks which led Bodenheimer to conclude that Shodan did not significantly affect PLC security [24]. This brings into question whether the attacks observed by Willhoit were targeted at the PLCs or just at the HMIs [21].

Regardless of Shodan's affect on targetability, there is no denying that these ICS attacks can have serious repercussions. The most well-known example of this is the 2010 Stuxnet worm, which exploited PLCs as part of an apparent attack on Iran's nuclear facilities. This worm caused physical damage to centrifuge devices within the facilities [2]. This attack was not a singularity, ICS threat modeling studies have shown that Denial of Service (DoS) and integrity attacks against chemical production PLCs can cause disruption or even explosions [3].

Attacks do not have to use malicious logic to cause damage. Project Basecamp, an experiment by Digital Bond, showed that an attacker could DoS, leak information, and modify the firmware of an Allen-Bradley PLC by simply using commands designed to improve ease of use [25].

From an outside perspective it would seem that traditional IT security principles could be applied to protect ICS devices. While some of these principles certainly could be applied, there are a few key differences between ICS and traditional IT systems that require consideration.

The first of these differences is apparent in the security goals of ICS systems. With traditional IT the importance of the elements in the CIA triad of confidentiality, integrity, availability follow directly in that order. ICS systems on the other hand value availability and integrity significantly higher than confidentiality [1]. The result is that ICS systems cannot simply be taken off-line to be patched and updated.

Likewise, often the industry resists updating software, because any minor change on these aging systems could cause an outage.

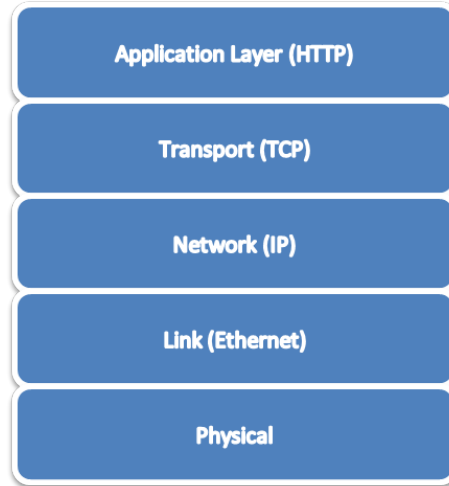
Aging systems are another issue in regards to ICS equipment. The typical life-cycle of an IT system is 5 years, while ICS systems often reach ages close to 20 years [1]. Due to this, ICS equipment does not have the same technological capabilities. Security techniques like encryption, which can be trivial on an IT network, can bring an ICS system to a halt. In fact typical ping sweeps, used by penetration testers, were shown to be responsible for the destruction of \$50K worth of electronic equipment [26].

### **2.2.3 Protocol Stack.**

To understand the variety of honeypot solutions available it is necessary to understand the underlying Internet protocol stack as shown in Figure 5. This stack explicitly outlines the different segments required to transmit information across traditional IT networks. Each segment is responsible for a different part of the transmission process.

The application layer is the most abstract of the segments. This segment is responsible for the application-specific communication being transmitted. The most common example is HTTP, which is responsible for delivering websites to computers. HTTP, as outlined in RFC 2616, has a series of defined messages [27]. These messages include the most commonly used GET request, which is used to retrieve information from websites. Both entities interested in communicating over HTTP must use these specifically-defined messages.

The transport layer is the next layer in the protocol stack. The best known transport protocol is TCP which is used for connection-oriented transmissions, where reliable delivery is required. This is because TCP includes acknowledgment/synchro-



**Figure 5. Internet Protocol stack**

nization techniques to alert the receiver that the information was received, and verify the correct order. For simpler messages that can be simply retransmitted if lost, UDP is often used due to its lower overhead.

This is all built upon the network layer which provides routing and logical addressing for messages it encapsulates into datagrams. The IP, as defined in RFC 791, is the most commonly utilized network layer [28]. This protocol contains a header that is comprised of data including the source and destination IP address. Using this information, connected hosts and routers can route traffic to its correct destination.

When emulating a device, all messages must follow these outlined specifications; if they deviate even slightly, communication will fail. This requirement makes creating honeypots to emulate these protocols extremely difficult. For this reason there are different honeypot solutions that emulate various layers of the protocol stack.

#### **2.2.4 Honeypots.**

*Honeypots* are extraneous systems designed to lure attackers into targeting them to provide an advantage to the network owner. These honeypots have an advantage over traditional intrusion detection systems (IDS) because no legitimate traffic should

be routed to them. This reduces false positives as any connection to the honeypot should be investigated. These systems, if implemented properly, should have no affect on production environment. The data recovered from these attempted intrusions can then be examined to best determine how to bolster defenses [4]. One example of the power of information gathered from honeypots can be seen in the “Kyoto 2006+” data set, which was intended to reveal the state of malware on the Internet. Researchers spent three years gathering Internet traffic and discovered that nearly half of all collected connections were attacks [7]. Even more serious is that about one percent of these attacks (425,719) contained unknown signatures that did not trigger the IDS.

Different categories of honeypots exist. These can be generally categorized into two overarching categories, HI and LI, based on three main of factors: fidelity, performance, and security [4]. Fidelity refers to the level of realism achieved, performance is the ability to handle sufficient amounts of traffic, and security concerns the threat posed to the integrated environment. All three of these factors vary in importance depending on the deployed environment thus require different solutions.

HI honeypots are typically resource-intensive honeypots that can range from full production systems to virtual emulated machines (e.g., VMware and Quick Emulator (QEMU)) [4]. These honeypots often contain the full range of capabilities of their intended emulation target, which means that they can often correctly implement the application layer of communication protocols, making it difficult for an adversary to realize that they are interacting with a honeypot. High interaction honeypots can pose an increased security risk to the integrated environment, because if compromised they can be used to pivot to production hardware.

LI honeypots on the other hand are lightweight programs that can be run on low-powered systems, such as the Raspberry Pi [4, 12]. These systems often only implement a small subset of the capabilities of their intended targets. Examples of LI



honeypots are the Deception Toolkit, LaBrea, Tiny Honeypot, GHH, and Php.Hop [29, 30, 31, 32, 33]. These systems all have different goals including malware capture, tarpitting (*slow down*), and adversary detection [4]. Regardless of their intended goal, the majority of LI honeypots implement only fully up to the transport layer, with secondary functions implementing various subparts of application layer logic. Hence, it is often easier for an adversary to ascertain when they are interacting with one of these systems.

### **2.2.5 Evaluation Tools.**

When evaluating the viability of a Honeypot a variety of tools can be used. The tools that are most relevant to this research are Nmap, Honeyscore, Wget, and each manufacturer’s PLC software.

#### **2.2.5.1 Network Mapper.**

Nmap is the de-facto standard for network discovery [34]. This tool sends the target various network packets to determine characteristics. Some of these packets are simple synchronize (SYN) requests to determine if a port is open/closed. Others such as those used for OS fingerprinting are a specifically-formatted sequence of packets. The response to these sequences is analyzed by Nmap and compared against a database of OS fingerprints, as shown in Figure 6. Each line of this fingerprint contains information on how the target responds to each type of request. Honeyd exploits this database to create personalities, since it now knows how to correctly respond in a way to trick Nmap into identifying the honeypots as their emulation target. As shown in Figure 7 the compilation of these scan results includes details such as open/closed ports, underlying OS, and device type.

```

Fingerprint ControlLogix 1756
Class general purpose
CPE cpe:/o:windriver:vxworks:5.5|
SEQ(SP=9B%GCD=1%ISR=A0%TI=I%CI=I%II=I%SS=S%TS=1)
OPS(O1=M5B4NW0NNT11%O2=M5B4NW0NNT11%O3=M5B4NW0NNT11%O4=M5B4NW0NNT11%O5=M5B4NW0NNT11%O6=M5B4NNT11)
WIN(W1=2000%W2=2000%W3=2000%W4=2000%W5=2000%W6=2000)
ECN(R=Y%DF=Y%T=40%W=2000%O=M5B4NW0%CC=N%Q=)
T1(R=Y%DF=Y%T=40%S=0%A=S+%F=AS%RD=0%Q=)
T2(R=N)
T3(R=N)
T4(R=Y%DF=N%T=40%W=2000%S=A%A=Z%F=R%O=%RD=0%Q=)
T5(R=Y%DF=N%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T6(R=Y%DF=N%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
U1(R=Y%DF=N%T=40%IPL=38%UN=0%RIPL=G%RID=G%RIPCK=Z%RUCK=0%RUD=G)
IE(R=Y%DFI=S%T=40%CD=S)

```

Figure 6. Nmap database example

```

Nmap scan report for 192.168.1.100
Host is up (0.0024s latency).
PORT      STATE SERVICE
80/tcp    open  http
44818/tcp  open  EtherNet/IP-2
MAC Address: 00:1D:9C:BE:67:3B (Rockwell Automation)
Warning: OSScan results may be unreliable because we
Device type: general purpose
Running: Wind River VxWorks
OS CPE: cpe:/o:windriver:vxworks
OS details: VxWorks
Network Distance: 1 hop

```

Figure 7. Scan of Allen-Bradley ControlLogix L55

#### 2.2.5.2 Honeyscore.

Honeyscore is a tool, created by Shodan's John Matherly, that can be used to deduce whether a system is real or a honeypot [35]. The tool is used by simply entering the IP address of the target into the text-box and clicking check for honeypot. The exact method of detection is not revealed as this would reduce the efficacy of the tool. The tool, as shown in Figure 8, claims to have extracted characteristics of known honeypots, and performs analysis on the supplied IP address to determine if the target exhibits any of these behaviors.

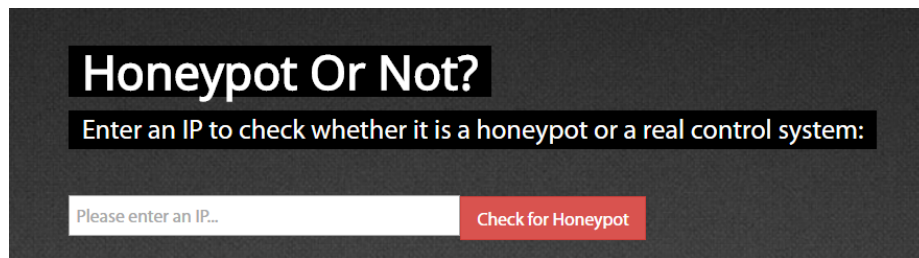


Figure 8. Honeyscore Interface

#### 2.2.5.3 Wget.

Wget, created in 1996, is a command-line program that retrieves content from web servers. It is part of the GNU project and supports the downloading over HTTP, HTTPS, and FTP protocols [36]. It is widely used among Unix users and is distributed with most Linux distributions. Designed for robustness and portability, the tool has the ability to recursively download linked resources, which is critical for dynamic web pages such as those on this research's PLC.

#### 2.2.5.4 RSLinx and STEP7.

RSLinx and STEP7 are tools, specific to the brand of PLC, that are responsible for displaying the PLCs information and applicable modules [37, 38]. Each tool uses a

different protocol (e.g., ENIP or ISO-TSAP) for communicating with the target PLC. These tools are utilized to connect to the PLCs and download/upload code needed for operation.

## 2.3 Related Research

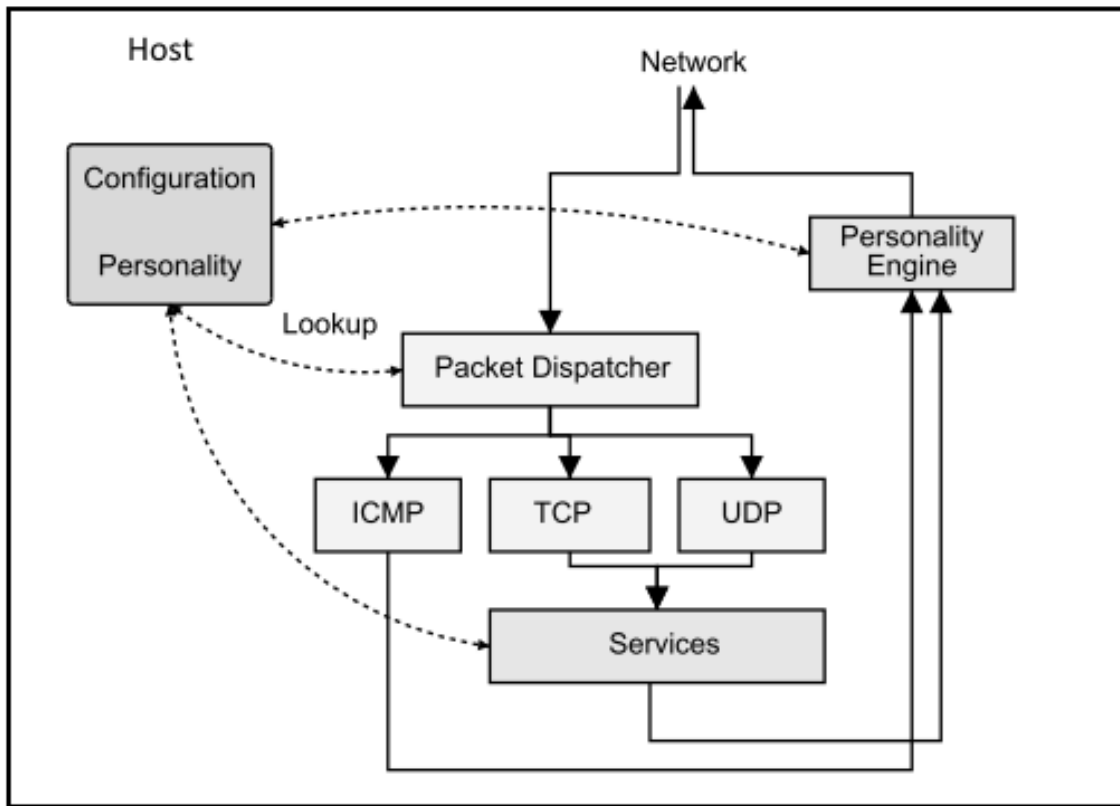
### 2.3.1 Honeyd.

Of all of the LI honeypot options, Honeyd is the most popular [8]. Honeyd itself is actually a framework for deploying a collection of honeypots. When creating a honeypot the configuration file must specify the emulation personality, the port specifications of the device, and various other options as shown in Figure 9. The emulation personality (e.g., Manual 1756) is chosen from a list included with the Honeyd installation. This personality tells the honeypot how to respond to trick the specially-formatted requests of Nmap. The port specifications (e.g., filtered or closed) define what action to take on all ports that are not otherwise specified. The other options include what subsystems to run, and the network and link-layer addresses for the honeypot. From this information Honeyd creates a series of honeypots that can be used to fill in the unused IP space on a network.

```
create default
set default default tcp action filtered
set default default udp action filtered
set default default icmp action closed
create rockwell
set rockwell personality "Manual 1756"
set rockwell default tcp action filtered
set rockwell default udp action filtered
set rockwell default icmp action closed
add rockwell subsystem "python /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/
ScriptGenEemulate.py /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/final_tree_port44818 44818 -
f -i eth0 -e $ipsrc"
add rockwell subsystem "python /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/
ScriptGenEemulate.py /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/final_tree_port80 80 -f -i
eth0 -e $ipsrc"
set rockwell ethernet "00:00:BC:E5:63:16"
bind 192.168.153.182 rockwell
```

Figure 9. Honeyd Configuration [1]

After configuration, Honeyd functions by designating the host computer to respond to Address Resolution Protocol (ARP) requests destined for the various Honeyd instance MAC addresses. As shown in Figure 10, once the traffic reaches the host machine it is routed to the packet dispatcher which performs a lookup to determine the matching Honeyd honeypot. Using this configuration file, Honeyd determines how to respond to Internet Control Message Protocol (ICMP), TCP, or UDP requests (e.g., RESET, SYN or IGNORE). Furthermore it determines if the target ports have any service scripts running, and if so transfers the data.



**Figure 10. Honeyd Architecture [1]**

The real power of Honeyd is the ability to create scripts to emulate services on individual ports, such as File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP). This allows Honeyd to bridge the application layer with the transport layer, and makes it a significantly more credible honeypot. Honeyd utilizes two main

types of scripts: regular and subsystem. Regular scripts are created and initialized for every separate connection to the honeypot. These are often reserved for lightweight scripts that do not require complex computation. A subsystem script is shared by all connections and only initialized during honeypot boot. This reduced overhead allows for more complex computation.

One of the biggest drawbacks to Honeyd is the need to create and maintain individual scripts for each application layer protocol. This process is extremely time consuming and can significantly increase the cost to benefit ratio of using Honeyd to an unfavorable level.

### **2.3.2 Automatic Protocol Emulation.**

To mitigate this major drawback of Honeyd, there have been various projects to automate the process of creating application-level emulators. Each of these projects achieve a high level of fidelity without relying on extensive configuration. These projects emulate the well-known protocols by observing port numbers and protocol traffic to choose from pre-configured scripts.

A “catering” honeypot created by Xiang and Ju called BAIT-TRAP is the first of these projects [39]. It observes traffic on the network and then can dynamically choose from a variety of honeypot scripts to create a relevant system from various physical or virtual honeypot shells. These shells are basic outlines of different systems (e.g., Linux and Windows) without specific running scripts. An example of this in action is initializing a honeypot running an HTTP daemon after seeing TCP SYN attempts to port 80. In this example, the shell could be Windows or Linux since both are legitimately capable of running web servers. The issue with the design is that the scripts must be created and stored in a database in order to be selected.

Although this project may simplify configuration of a series of honeypot instances, it does nothing to reduce the complexity of creating protocol-dependent scripts.

Chowdhory et al. extended this idea by using a modified version of data mining they call *service mining* [40]. Their system used an initial database of arguments and keywords to break down network traffic into related sets of keywords. Using these keywords they were able to successfully emulate the FTP protocol.

Similar work was done in automatically emulating the web interface of PLC's, using TCP and HTTP. Fink used Wget and tcpdump to extract information, and then reconstructed the TCP data. Using this extracted information Fink was able to mimic the web interface of an arbitrary PLC [41].

### **2.3.3 Advanced Honeypots With Replay.**

To create truly flexible honeypots capable of emulating application-layer traffic, applicable protocols must be completely reverse engineered. After the protocol is understood it can then be replayed to accurately simulate an ICS device.

#### **2.3.3.1 Protocol-Agnostic Replay.**

RolePlayer is a tool designed to listen to communication between two devices, and then using a series of algorithms; it is able replay this communication to emulate a participant in the conversation [42]. It is also able to act as a proxy to the real device, so that if it encounters an unknown message it can query the real device to decide how to respond. This tool can run as a proxy to a Honeyd instance but both tools must be running simultaneously.

ScriptGen is a framework that goes one step further by creating Honeyd scripts based on captured traffic [10]. After creation, these scripts can be simply included in the Honeyd configuration. ScriptGen works by using a series of algorithms to create

a state machine. Using this state machine, the created scripts can emulate a device by matching traffic to its state machine, and iterating over the sequential nodes. An updated version of ScriptGen adds the proxy capability of Roleplayer, and the ability to detect protocol dependencies [43].

ScriptGenE extends the functionality of ScriptGen by implementing ICS-specific features, such as default error messages and unknown transition handling [11]. This framework was designed by Warner to be protocol agnostic to better emulate PLC's which often use proprietary protocols. This framework creates protocol trees which then need to be integrated into Honeyd scripts or subsystems.

### **2.3.3.2 Protocol Agnostic Honeypots.**

GQ is a malware execution “farm” that utilizes Roleplayer as a LI honeypot front end. It was initially designed for tracking worm propagation [13, 44]. GQ uses a centralized gateway to create various infected VMs and then tracks these systems using a VM monitor. This monitor makes containment decisions to help control and regulate the spread of malware within the “farm”.

SGNET is similar to GQ in that it was designed to study malware, except it focuses on “code injection attacks”. SGNET utilizes a distributed architecture and relies on ScriptGen for its LI front end, and QEMU emulators for the HI back end [14]. This combination allows the framework to evaluate all parts of a code injection attack (i.e., exploit, control hijack and payload execution). An improvement to this framework called Mozzie enhances the project by using ScriptGen on both ends of malware traffic [45]. This improvement allows ScriptGen to generate both sides of traffic (i.e., client and malware) allowing traffic to be studied without having to communicate out to the external web.



AWESOME, or Automated Web Emulation for Secure Operation of a Malware-Analysis Environment, combines many of the above techniques into one framework. It uses Honeyd and Argos, a QEMU based network traffic monitor, for its LI and HI honeypots respectively and employs ScriptGen for communication replay. The purpose of this system is to capture and analyze malware. [46]

#### **2.3.4 ICS Honeypots.**

Honeypot solutions are traditionally used to emulate traditional IT infrastructure; applying this technology to ICS networks can be difficult. However, due to the growing concern for the security of ICS equipment, there have been a few attempts to create solutions designed for the ICS environment. These solutions, while effective, are often specific to certain protocols and difficult to correctly implement.

The first of these solutions is the open source honeypot Conpot created by the HoneyNet Project [47]. This LI honeypot implements the Modbus and Simple Network Management Protocol (SNMP) protocols. The default configuration simulates a basic Siemens S7-200 PLC with a CPP431 module. It was designed to be easily customizable, meaning that if the same protocols were used it could be configured to resemble another PLC type. However it still requires significant effort to accurately mimic another set of protocols.

Another ICS honeypot solution is an AFIT project called Honeyd+ [12]. It is an extension of Honeyd designed to be deployed on a Raspberry Pi. As shown in Figure 11 the project is intended to allow multiple geographically-separate instances of Honeyd+ to proxy to a single PLC. Also, the project focused on improving Honeyd by removing MAC and IP discrepancies in the PLC web interfaces by implementing a search and replace function. This prevents the attacker from ascertaining he is interacting with a honeypot.

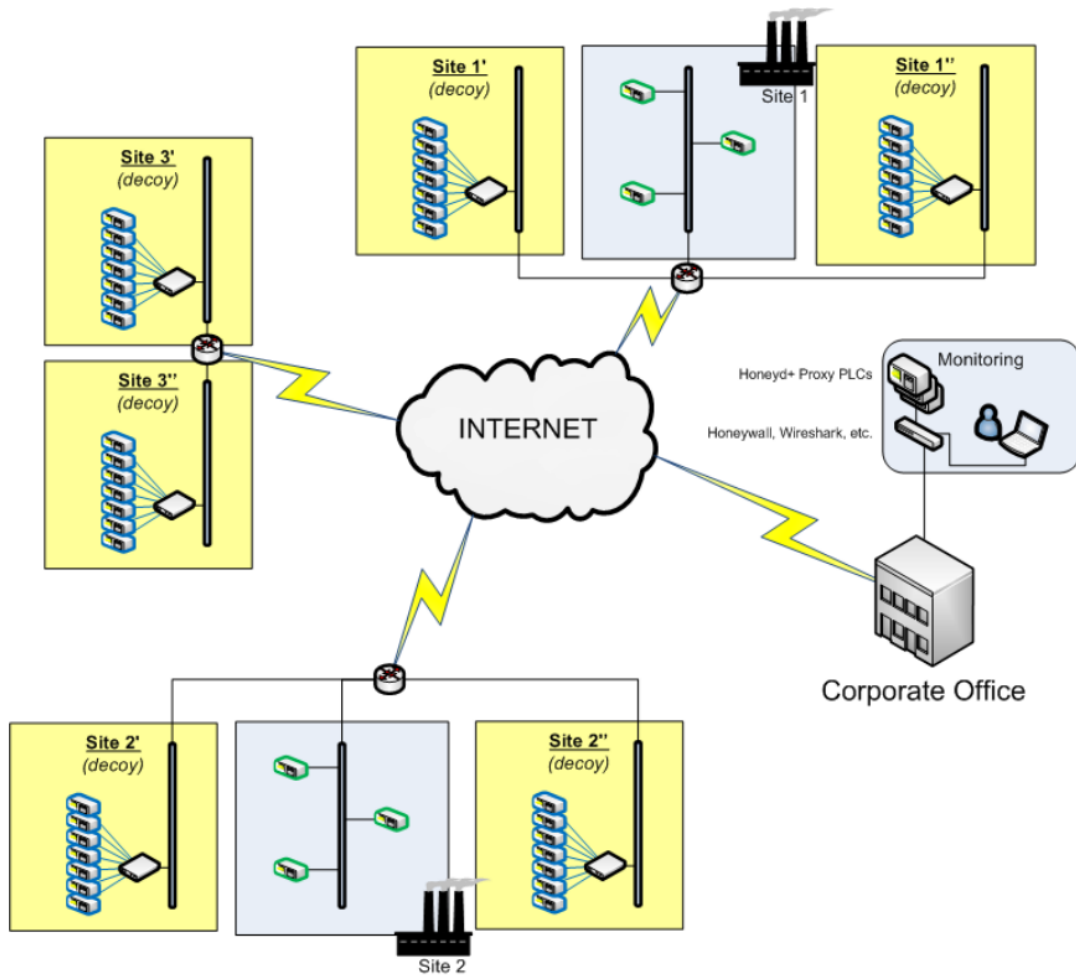


Figure 11. Example Honeyd+ Production Configuration [12]

Digital Bond created SCADA Honeynet, a freely-available ICS honeypot solution that consists of two VMs. One VM acts as an Intrusion Detection System (IDS) to monitor the other VM which emulates a PLC running Modbus, Telnet, SNMP, and HTTP protocols. The IDS uses Digital Bond's specialized Quickdraw IDS signatures to detect malicious attacks against the second VM. The user has the option to replace the simulated PLC with an actual PLC to retain the monitoring capabilities while improving authenticity [48].

CryPLH is a stripped down Ubuntu VM intended to emulate a PLC [49]. This honeypot uses built in iptables to create the appearance of being a Siemens Simatic 300. A series of scripts then emulate the HTTP, SNMP, and ISO-TASP protocols. This model suffers because each service must be individually configured.

Jaromin created another honeypot that uses a stripped down version of Linux. He was able to emulate a Koyo DirectLogic 405, utilizing HTTP and Modbus protocols, on a Gumstick device [50]. This project works well within its scope, but can only emulate the specifically chosen PLC.

## **2.4 Chapter Summary**

This chapter discusses the reliance of critical infrastructure on ICS equipment. This equipment was designed with priorities that differ from traditional IT systems. However to achieve efficiency goals, ICS devices have been integrated into traditional IT networks. This integration has created security concerns because if not correctly configured these devices can be targeted by attackers.

To help protect against these attacks, various honeypot solutions have been applied to ICS networks. Each of these solutions has various pitfalls, including small protocol emulation portfolios and extensive configuration requirements. These pitfalls

highlight the need for a ICS specific honeypot that is easily configured to emulate the vast variety of ICS protocols.

## III. Framework Integration

### 3.1 Overview

This chapter outlines the integration of an automatic application-layer emulator with the Honeyd honeypot framework. This integration requires changes to both components in order to achieve the desired functionality.

### 3.2 Motivation and Application

Previous research demonstrated that ScriptGenE was a capable application-layer emulator. However in order to be a credible honeypot, it was clear that network-layer emulation was required. To accomplish this goal it was determined that integration with Honeyd was appropriate.

### 3.3 The ScriptGenE Framework

ScriptGenE is an extension of the ideas detailed by Corrado Leita [10]. It was developed in Python, and consists of approximately 5700 lines of code (not including third party libraries) [11]. The p-tree is the center of the framework with various scripts responsible for its generation, manipulation, and replay.

#### 3.3.1 Framework Overview.

The ScriptGenE framework consists of a series of Python files:

- ProcessPcap.py - Responsible for pulling TCP data out of Packet Capture (PCAP) files.
- ScriptGenE.py - Builds initial and generalized p-trees.

- GeneralizeTree.py - Generalizes initial p-trees built by ScriptGenE.py.
- CombineGtrees.py - Combines p-trees at their root.
- ScriptGenEemulate.py - Loads p-trees and replays server messages in context during conversations with clients messages.

However as shown in Figure 12, the user only interacts with two of these files: ScriptGenE.py and ScriptGenEemulate.py.

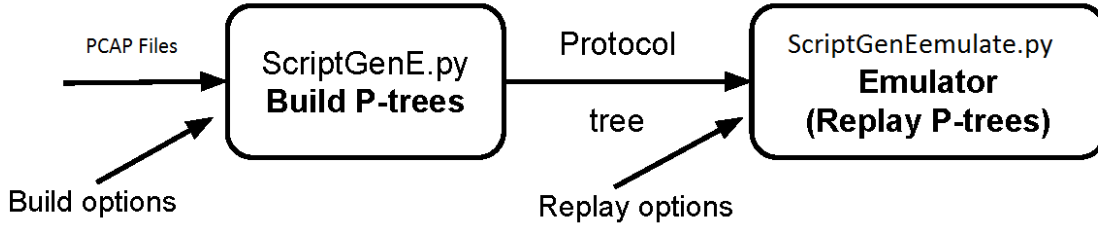
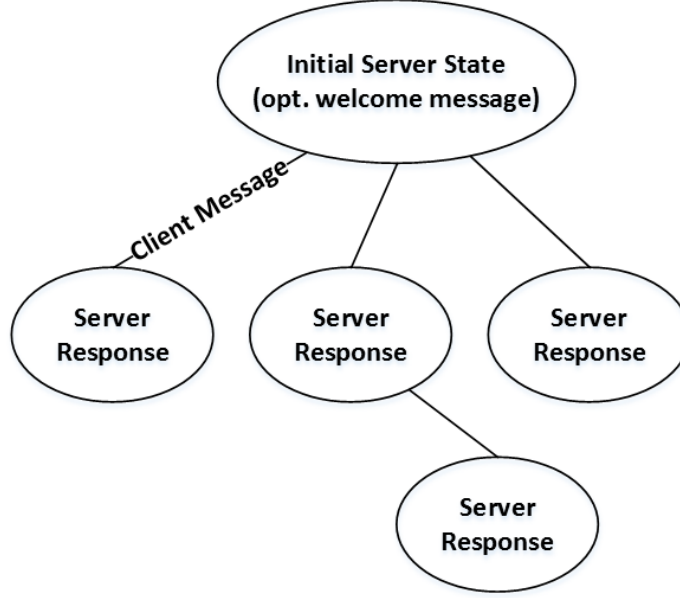


Figure 12. ScriptGenE Framework Overview

### 3.3.2 ScriptGenE.py.

ScriptGenE.py is responsible for creating the p-tree. This process begins with a set of PCAPs consisting of a particular protocol conversation. Every complete connection is then separated into a discrete tree. This tree is constructed by utilizing TCP sequence and acknowledgment numbers to designate message order. As shown in Figure 13, the nodes in the tree represent server messages, while the edges correspond to client messages. These initial trees are then exported as Python pickle files, which are serializations of the Python tree object hierarchy [51]. For reference, Figures 14 and 15 show example segments of the ENIP and HTTP protocol trees.

After the initialization step, ScriptGenE.py calls GeneralizeTree.py to combine the initial trees. This is done by first consolidating every duplicate edge that has matching



**Figure 13. Protocol Tree Structure**

parent nodes and client data. Then a second round of consolidation combines messages that are semantically equivalent. Semantic equivalence is decided through the use of various macro/micro clustering algorithms. These algorithms are built upon the Protocol Informatics (PI) project, which uses bioinformatic based algorithms (e.g., Smith-Waterman and Needleman-Wunsch) to sequence protocol messages [52]. These edges are then represented by regular expressions (regex) that encompass the characteristics of the combined messages.

Consolidation also locates environmental fields (e.g., IP addresses) and replaces them with markers so that they can be replaced with the accurate details during replay. Finally a default error is created, and the p-tree is exported to a Python pickle file for replay in ScriptGenEemulate.py. This research did not focus on p-tree creation, so this is just a brief overview of a complex process. For more information on how p-trees are built and semantically combined see Warner’s description of the ScriptGenE.py process [11].





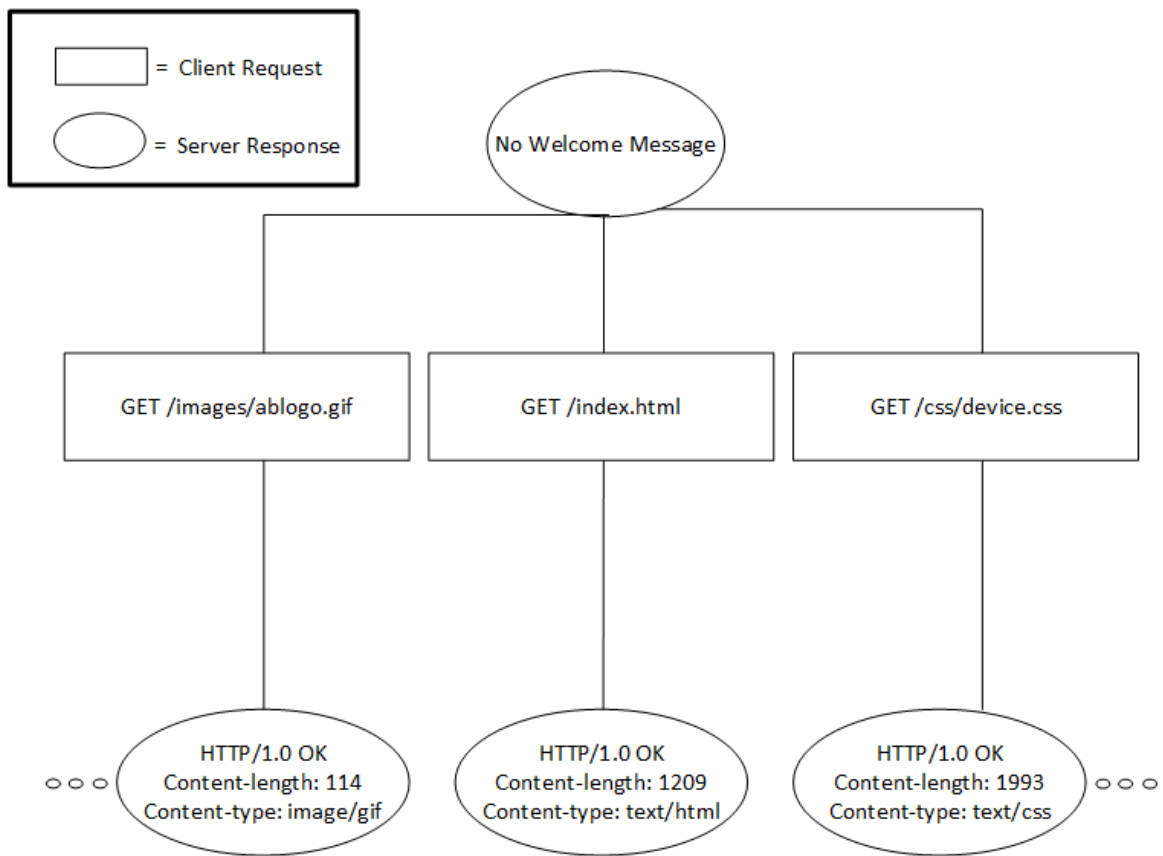


Figure 15. Example Segment of HTTP Protocol Tree

### 3.3.3 ScriptGenEemulate.py.

ScriptGenEemulate.py is the emulator script in the ScriptGenE framework responsible for the replay of the p-tree. The process begins by loading a Python pickle file consisting of the p-tree for a particular protocol. A socket is then opened for the client to connect. Starting at the root of the p-tree, ScriptGenEemulate.py matches incoming client requests to the edges extending out of the current node. If a match is found, the context switches to the next node following that matched edge. This next node represents the next server message to be sent to the client.

If a match is not found within the current nodes edges, then a unknown transition is encountered. This is handled by allowing backtracking from the current nodes context. First, only previously traversed edges are considered, this is to increase efficiency in polling protocols often utilized by PLCs. If a match is still not found, then the entire p-tree is traversed in search of a match. If this still does not result in a match then the default message is sent to the client.

The ScriptGenE framework is designed to be run from the Linux command line. As shown in Figure 16, the command line options are extensive. Many of the options are related to the proxy mechanism of the framework. This allows unknown messages to be sent to a legitimate back-end PLC for the correct response. This research does not utilize this capability, choosing to instead focus on improving viability as a standalone PLC emulator.

## 3.4 ScriptGenE Framework Changes

While the primary effort of this research focuses on integrating the two previously discussed frameworks, effort is also made to further improve the emulation of the underlying ScriptGenE framework.

```
usage: ScriptGenEemulate.py [-h] [--host HOST] [-f] [-c CONNECTIONS]
                             [-o {client,server,all}] [-r {never,open,always}]
                             [-t TARGET] [-p PROXY_PORT]
                             [-m {catchup,min_catchup,lockstep,latelock,min_latelock,t
                             [-s [n]] [-d [DEFAULT_ERROR]] [-e EMULATION]
                             [-u UDPSOCKET] [--debug | -v]
                             tree_file port
```

ScriptGenEemulate version 0.2 by Justin Gallenstein. Extended from ScriptGenEreplay version 0.1 by Kyle Girtz and Phillip Warner. This Python script imports a generalized protocol tree from ScriptGenE output and replays server responses based on client request matches. Proxy functionality allows emulation of gaps in the protocol tree.

#### positional arguments:

```
tree_file      Input tree filename with path (no extension)
port           Server port to accept connections on [1-65535]
```

#### optional arguments:

```
-h, --help            show this help message and exit
--host HOST           Host name of target (e.g. www.example.com)
-f, --forever         Run server forever until Keyboard Interrupt (Default:
                        Off)
-c CONNECTIONS, --connections CONNECTIONS
                        Number of connections to accept if -f not used
                        (Default: 1)
-o {client,server,all}, --original_data {client,server,all}
                        Use original data instead of environmental data for
                        client, server, or both (Default: off)
-r {never,open,always}, --repeat {never,open,always}
                        Repeat (backtrack) after end of tree reached? Open
                        means repeat when final node not RST or FIN (Default:
                        open)
-t TARGET, --target TARGET
                        IP address of proxy target. Proxy is disabled if
                        address is not provided or invalid.
-p PROXY_PORT, --proxy_port PROXY_PORT
                        Port of proxy target service [1-65535] (Default: same
                        as server port)
-m {catchup,min_catchup,lockstep,latelock,min_latelock,templock,min_templock,triggerlock,templock,min_templock,triggerlock,min_triggerlock}
                        Determines algorithm for catching up proxy target to
                        current context for a particular request. "min_"
                        prefix uses quick proxy synchronization with minimal
                        tree replay (Default: lockstep)
-s [n], --strict [n] Minimum number, n, of successful client msgs required
                        before backtracking through tree. This takes
                        precedence over the repeat setting. (Defaults: No
                        restriction (n = 0) if option not used. No
                        backtracking allowed if n = -1)
-d [DEFAULT_ERROR], --default_error [DEFAULT_ERROR]
                        Override tree's default error msg with file name -or-
                        byte stream (e.g. '\xaa\xbb') to use as default error
                        message. Environmental link tags (e.g.
                        '#E-server_ip#') will be replaced by corresponding
                        info. Use '#REPEAT#' or leave blank to simply repeat
                        the last server message (Default: Use the one defined
                        in tree_file)
-e EMULATION, --emulation EMULATION
                        IP address of honeypot. If using Honeyd use $ipsrc
-u UDPSOCKET, --udpsocket UDPSOCKET
```

Figure 16. ScriptGenEemulate.py Usage

### 3.4.1 UDP Support.

As mentioned in Section 3.3.2, the ScriptGenE.py tree building process utilizes TCP sequence/acknowledgment numbers to construct the p-tree. This allows for quicker responses as each child node designates the next expected server message that needs to be sent. This negates the need to always perform a complete tree search.

This research expands this functionality by adding UDP support to the framework. A new argument `-U` is added to ScriptGenEemulate.py to signify that the script should open up a UDP socket in place of the usual TCP. As discussed in Section 2.2.3, UDP messages are lightweight and do not include information used to construct the organized TCP p-tree. Instead a separate UDP tree with a depth of one is created. Using the algorithm described below, each edge, representing a possible client message match, is checked to find the correct server message response.

### 3.4.2 Improved Backtracking Algorithm.

In the previous ScriptGenE implementation if the protocol tree match was not exact to a designated regex or string, then either a default error was sent to the client or the request was proxied to the legitimate back-end PLC. This is a significant shortfall because it is likely a large portion of ScriptGenE implementations will not have a back-end PLC. A simple example of this use-case in action is shown with a typical GET request to a PLC's web interface. Shown below is the GET request as stored in a p-tree as well as a GET request for the same object.

1. Protocol Tree: GET /index.html User-agent: Mozilla/4.0 (compatible MSIE 9.0; Windows NT 6.1)
2. Client request: GET /index.html User-agent: Mozilla/5.0 (IE 11.0; Windows NT 6.3)

This client request should be considered a match to the stored p-tree request, since they are both requesting the `index.html` resource. However, since the previous algorithm did a simple Python `str.find` to see if all characters matched, these would not be considered a match, and the client would be presented with a default error. The new algorithm goes about solving this problem in a protocol agnostic way consistent with ScriptGenE’s design. The problem could easily be solved for the HTTP protocol with a non-agnostic solution by simply parsing the string after the GET to find a match. However, this research is designed to solve it in a protocol agnostic way.

The new algorithm is implemented within the unknown transition functionality of ScriptGenE. This functionality is only called when a match cannot be found in the protocol tree within the current node’s outgoing edges. This new algorithm considers all edges in the protocol tree, and applies the algorithm described below. Implementing it within the unknown transition functionality reduces the negative impact on the emulator’s speed and efficiency.

The first step before running the algorithm is the generation of a “heatmap” of the protocol. This heatmap is created on honeypot startup and involves comparing all of the edges of the p-tree. Using these comparisons a mapping is created to show where the variation occurs. This allows the algorithm to score the edges more accurately. A match within a region in which all of the p-tree edges are very similar should be scored lower than a match in a diverse region. An example heatmap, for the HTTP tree, is shown in Figure 17. The darker the red, the more similar the region. The darkest regions of the heatmap are the GET header and User-Agent region, this is because all of the requests are GET requests from the same computer (i.e., same User-Agent). Not all sections of the requests are the same length so boundary values

tend to fade into the next regional color. This color is represented in the algorithm with a value between 0 - 1.

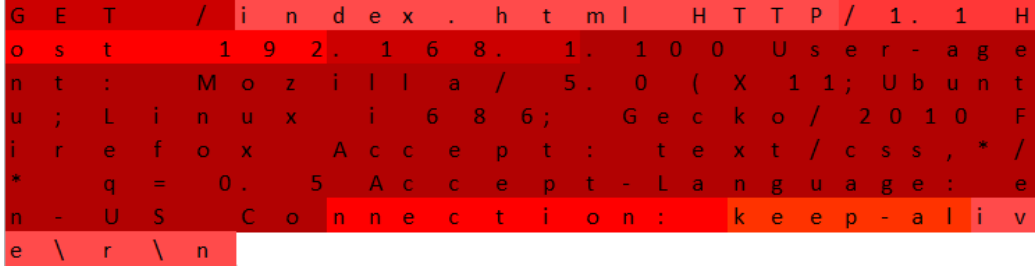


Figure 17. Heatmap Example

The actual matching algorithm is built upon the “gestalt pattern matching” method published in the late 1980’s by Ratcliff and Obershelp [53]. This algorithm focuses on finding the longest contiguous matching subsequence and then applying this idea recursively to the left and right of this subsequence. After this stage the algorithm has a series of matching “blocks” that the two sequences have in common. An example of this is shown in Figure 18, the input sequences are the two GET requests shown above. The first two items in each match block (i.e.,  $ab$ ) designate the index of the matching substring in each string respectively, while the size is simply the length of the string.

The equation to compute the total match score between a protocol tree edge and client request is

$$score = \sum_{i=0}^n (size_i * a_i / b_i) * heatmap_{ab} \quad (1)$$

where  $n$  is the total number of match blocks,  $size_i$  is the size of the match string,  $a_i$  is the index of the match in the first string,  $b_i$  is the index of the match in the second string, and  $heatmap_{ab}$  is the heatmap score of both indices.

The quotient multiplier is utilized to give preference to matches that exist closer together in their overall string. This is because even if a match is found, its appli-

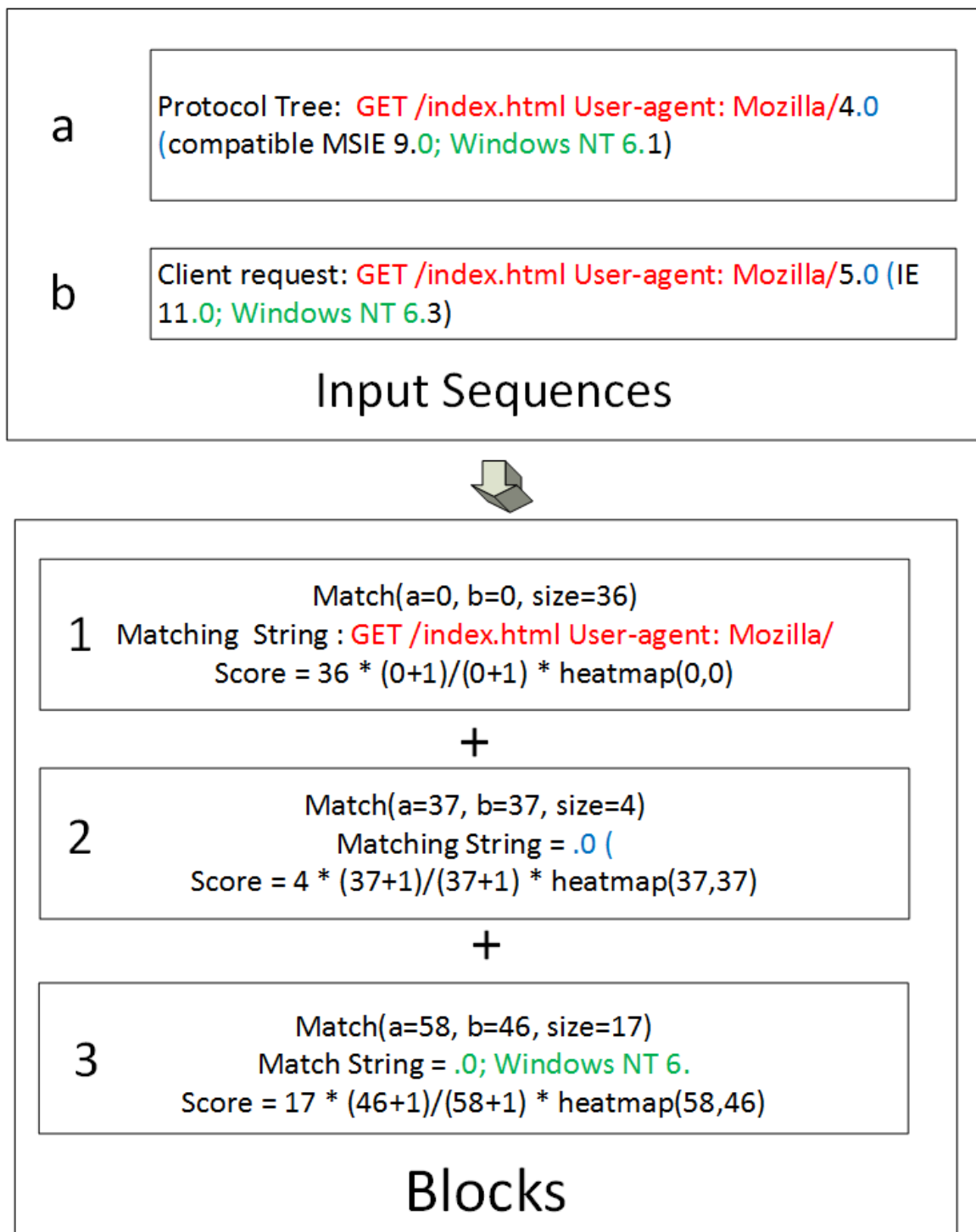


Figure 18. Matching Algorithm Example

cability to the similarity of the two requests is significantly lower if they are located in vastly different regions. When calculating the quotient the smaller index is always the dividend to ensure larger gaps are scored as penalties. Also since the index starts at zero, one is always added to both indices to avoid divide by zero errors.

This result of this equation is then stored as the score for that edge, this process continues until all edges are considered and a max score is found.

### **3.4.3 Automatic Creation of Honeyd Profile.**

As discussed in Section 2.3.1, the Honeyd framework relies on a configuration file to create an emulated device. This file consists of an OS personality, port specifications, MAC address, and designated services. This project was developed to automate the creation of emulated PLCs. In order to accomplish this, Fingerprint.py was written to automatically configure all of these options. This approximately 200-line Python script creates a matching configuration file for a given target IP address. This script utilizes an Nmap subprocess to scan the target IP address and uses the scan results to set ports, designated services, MAC and IP addresses, and a new matching personality to add to the Nmap database. This personality must be created because the database of personalities is preset and defined by the Nmap version installed. It includes well-known personality variants such as Windows, Linux, and Mac but often does not include matches for PLC emulation.

### **3.4.4 ScriptGenEemulate.py Alterations.**

In order to integrate ScriptGenE into the Honeyd framework, a few changes are made. Previously `-i interface` was used as an argument to determine the IP address of the ScriptGenEemulate.py instance. Integration with Honeyd requires the ability to initialize each ScriptGenEemulate.py instance based on its applicable hon-



eypot. To accomplish this a new `-e ipaddress` argument is added. When utilizing `ScriptGenEemulate.py` within the Honeyd configuration file the value supplied to this argument is `-e $ipsrc`. This serves as a placeholder that Honeyd replaces with the IP address of whatever honeypot is applicable. This ensures that all traffic originates from the appropriate PLC in the eyes of the attacker.

Also as shown in Figure 19, previously within `ScriptGenEemulate.py` multiple options were set upon socket initialization. These are removed as they were causing catastrophic errors when integrating with Honeyd. Honeyd is set up to hook into socket calls, and intercept requests before being sent to the kernel. These initialization options are not supported and cause segmentation faults during execution.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Creates TCP socket for client to connect to.
s.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1) # Turn off nagling so that small TCP packets are not combined.
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # Prevent errors when a program is killed and leaves open ports.
s.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1) # Turn on Keep alive packets to keep the client interested .
s.setsockopt(socket.SOL_TCP, socket.TCP_KEEPIDLE, 8) # Wait 8 sec of idle time before sending KA.
s.setsockopt(socket.SOL_TCP, socket.TCP_KEEPIVTL, 8) # 8 sec between KA.
s.setsockopt(socket.SOL_TCP, socket.TCP_KEEPCNT, 5) # Declare the client dead if KEEPCNT unanswered KAs reached.
```

**Figure 19. Removed ScriptGenEemulate.py Socket Options**

Removal of those options caused problems with `ScriptGenEemulate` functionality. Previously, the option to prevent combining of small TCP packets was enabled. Removal caused the socket to combine many of the smaller messages into one large TCP packet. This packet would not match any of the edges in the p-tree because in reality it is multiple packets. This problem required reworking the logic of `ScriptGenEemulate.py` to allow for packet dissection and storage for future matching.

As shown in Figure 20, the framework also had segments of code written to handle aspects of different protocols. To get `RSLinux` to recognize the emulated target, this flag had to be set to 1 (default is -1) in the code for it to perform a set of actions. This went against the protocol-agnostic goals of the framework, so these are removed in favor of protocol-agnostic solutions.

```

# TODO / switch to individual messages during tree build or simply
if msg_number == -1: # ENABLE split for first message w/ val of 1
    msg1 = msg[0 : 66]
    msg2 = msg[66 : 66+50]
    msg3 = msg[66+50 : ]

```

**Figure 20. Non-Protocol Agnostic Code Required to Emulate ENIP Protocol**

### 3.4.5 Honeyd Integration.

Integration with Honeyd, created by Neil Provos in 2003, presents a few technical hurdles. This software was last officially updated in 2007 (version 1.5c), prior to Nmap updating its OS detection algorithm. The ability of Honeyd to emulate a chosen target relies on this detection algorithm, and thus this version was almost obsolete. In 2011 a technology company called Datasoft wanted to utilize the Honeyd functionality within a product, so they created Honeyd version 1.6d [54]. This new version contained logic to handle the new Nmap OS detection method.

However, outside the scope of their project’s requirements, this new release was not tested thoroughly for the effect of these updates on the original framework. This led to segmentation faults when operating within this research’s scope, as ScriptGenE is implemented as a Honeyd (version 1.6d) subsystem. As discussed in Section 2.2.4, this allows the honeypot to share the one Python script for multiple connections, instead of creating a new instance with each connection. These segmentation faults are fixed by analyzing the code and correcting the functionality. One example of these changes is shown in Figure 21, with the original code commented out. This fix required that the pointer to the event was appropriately assigned. Previously the event object was created, but it was not assigned to the connection (i.e.,  $tmp \rightarrow ev$ ). Therefore, when a connection was added to the queue with the `event_add` method it was pointing to a NULL object.

```

//TRACE(port->sub_fd,
//      event_new(libevent_base, port->sub_fd, EV_WRITE, cmd_subsystem_connect_cb, tmp));

//TRACE(event_get_fd(tmp->ev),
//      event_add(tmp->ev, NULL));

struct event *e= event_new(libevent_base, port->sub_fd, EV_WRITE, cmd_subsystem_connect_cb, tmp);
tmp->ev=e;
event_add(tmp->ev, NULL);

```

**Figure 21. Fixed Honeyd Code Example**

Similarly, bugs are fixed in the way Honeyd sets TCP window size. As shown in Figure 22, regardless of negotiation the window size was always initialized to 0. This led to errors when transmitting data to the application layer as it was unable to receive messages in one segment (i.e., one message would be divided into multiple separate packets). This lead to failed attempts to generate p-tree matches on partial data packets. To fix this issue the code was updated throughout to initialize the window size to 65535, which is the maximum standard window size for TCP.

192.168.159.182	192.168.159.1	TCP	54	80 → 16518 [SYN, ACK] Seq=0 Ack=1 <b>Win=0</b> Len=0
192.168.159.1	192.168.159.182	TCP	54	16518 → 80 [ACK] Seq=1 Ack=1 Win=65392 Len=0
192.168.159.1	192.168.159.182	TCP	55	<b>[TCP ZeroWindowProbe]</b> [TCP segment of a reassembled PDU]

**Figure 22. Zero Window Error**

### 3.5 Design Summary

New features within the ScriptGenE framework include:

1. Fingerprint.py script for automatically configuring a Honeyd instance of a PLC.
2. New algorithm to select best-matching request in protocol tree.
3. Implemented UDP packet support.
4. ScriptGenEemulate.py changes to allow for integration with Honeyd.

5. Fixes to Honeyd 1.6d to allow for integration.

## IV. Research Methodology

### 4.1 Goals

This research focuses on further developing a framework capable of automatically configuring PLC honeypots. The following questions related to the extensions made to ScriptGenE are addressed:

1. Can the emulator accurately choose best-fit responses when exact matches are not available?
2. Can the integration with Honeyd fool detection/industry tools and emulate the industrial network of a simulated prison?

### 4.2 Approach

For this research a small-scale prison network, utilized for training cyber first-responders, is selected to be emulated. Developed by Joseph Daoud, this network is shown in Figures 23 and 24 and consists of three PLCs responsible for controlling a series of locks, all controlled by a centralized HMI [55]. This network is chosen because successful emulation would demonstrate the ability to simulate a small network with a series of honeypots.

To begin experimentation first PCAP files are generated by utilizing Wireshark to record network traffic between the legitimate PLCs and the tools described in Section 4.6.3. These tools are chosen because they are representative of traffic generated by an attacker in the early reconnaissance phase. After generating and capturing the required protocol traffic with the legitimate PLCs, the PCAPs are used by ScriptGenE.py to build p-trees as discussed in Section 3.3.2.

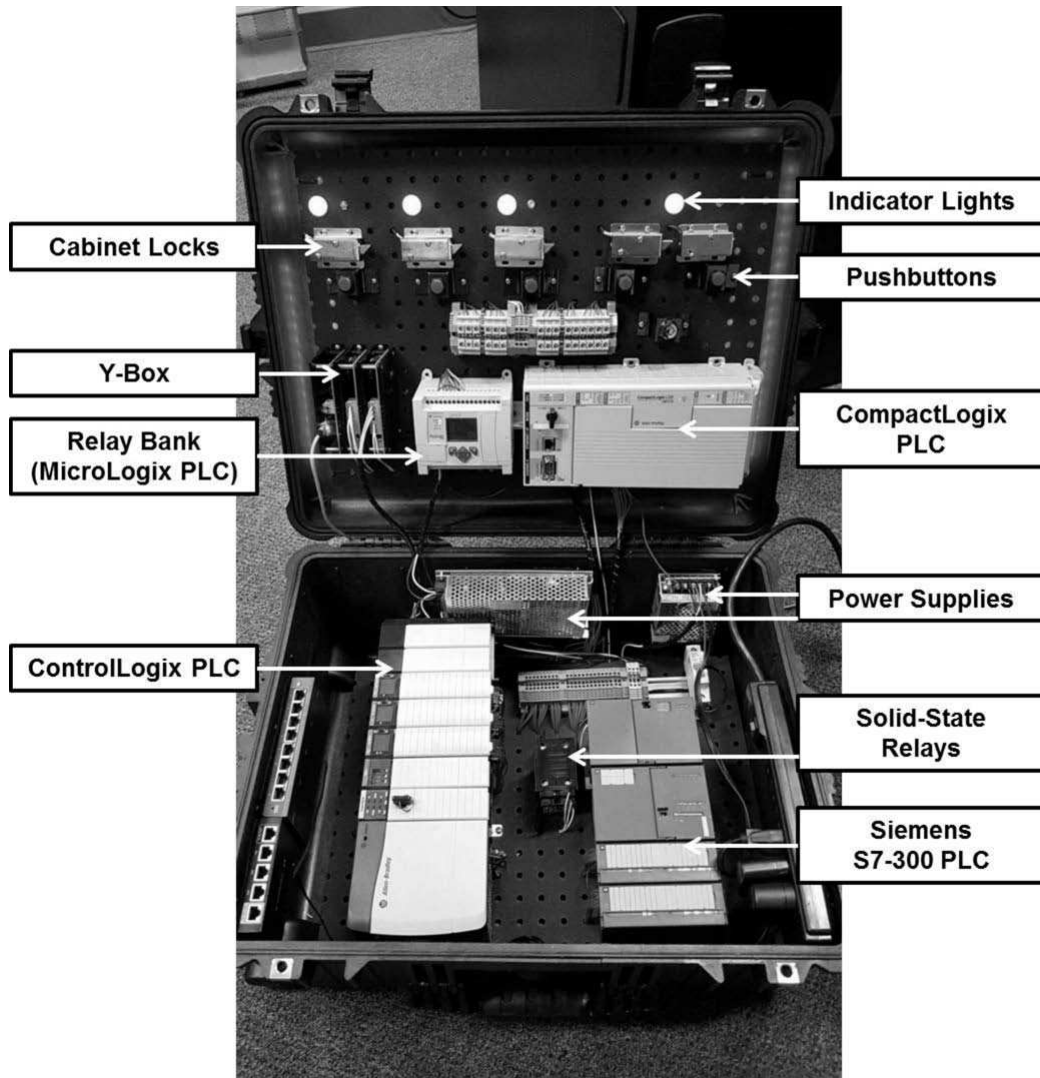


Figure 23. Training Prison Network [55]

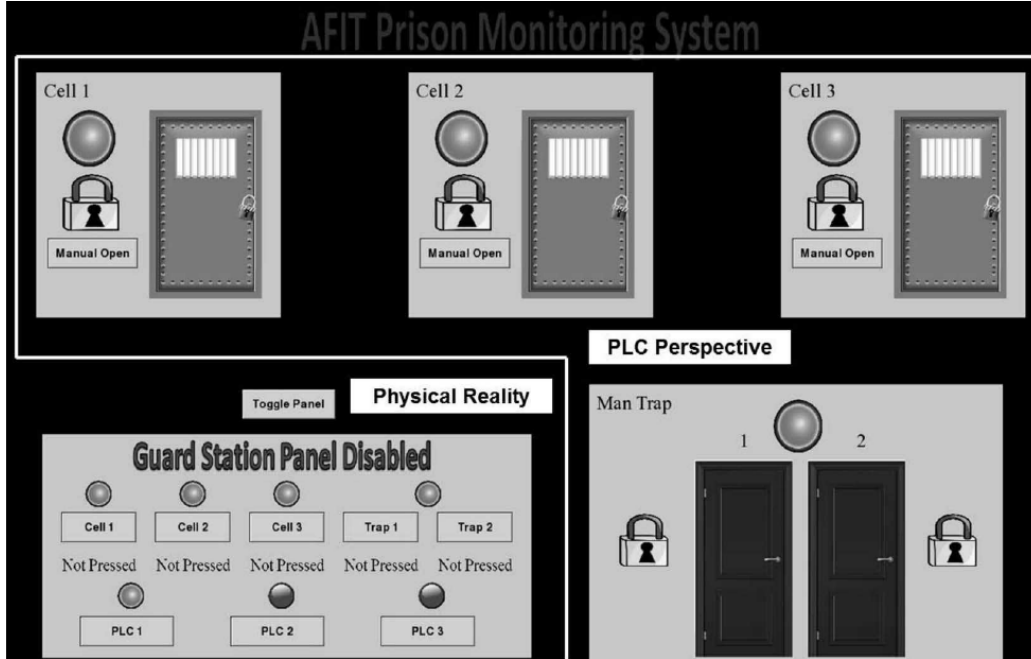


Figure 24. Training Prison Network HMI [55]

Then as the finalized Honeyd configure file shows in Figure 25, these p-trees are supplied to ScriptGenEmulate.py as respective inputs for each port. Now when Honeyd is initialized it creates three network layer honeypots (i.e., rockwell\_compact, rockwell\_control, and siemens\_s7300) with these ScriptGenEmulate.py instances listening on ports to simulate application layer functionality. The Allen-Bradley ControlLogix has two ScriptGenEmulate.py subsystems, as one is responsible for running its web server. These honeypots are further configured to imitate the legitimate PLCs by utilizing Fingerprint.py to import Nmap personalities (i.e., Allen Bradley CompactLogix, Allen Bradley ControlLogix and Siemens s7-300 ) into Honeyd that match each desired imitation target. Each honeypot is designated to have its own assigned IP address, and a MAC address that corresponds to the PLC manufacturers' assigned address block. They are also all set up to respond to all TCP, UDP, and ICMP messages with filtered, or closed responses respectively. Note this only applies to ports that are not otherwise designated to host a subsystem.

```

create default
set default default tcp action filtered
set default default udp action filtered
set default default icmp action closed

create rockwell_compact
set rockwell_compact personality "Allen Bradley CompactLogix"
set rockwell_compact default tcp action filtered
set rockwell_compact default udp action filtered
set rockwell_compact default icmp action closed
add rockwell_compact subsystem "python /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/ScriptGenEemulate.py /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/Justin\ pcaps/ENIP/L23E/final_tree_port44818 44818 -f -i eth0 -e $ipsrc"
set rockwell_compact ethernet "00:00:BC:E5:63:16"
bind 192.168.159.201 rockwell_compact

create rockwell_control
set rockwell_control personality "Allen Bradley ControlLogix"
set rockwell_control default tcp action filtered
set rockwell_control default udp action filtered
set rockwell_control default icmp action closed
add rockwell_control subsystem "python /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/ScriptGenEemulate.py /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/Justin\ pcaps/ENIP/ControlLogix/final_tree_port44818 44818 -f -i eth0 -e $ipsrc"
add rockwell_control subsystem "python /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/ScriptGenEemulate.py /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/Justin\ pcaps/ENIP/ControlLogix/final_tree_port80 80 -f -i eth0 -e $ipsrc"
set rockwell_control ethernet "00:1D:9C:BE:67:51"
bind 192.168.159.200 rockwell_control

create siemens_s7300
set siemens_s7300 personality "Siemens s7-300"
set siemens_s7300 default tcp action filtered
set siemens_s7300 default udp action filtered
set siemens_s7300 default icmp action closed
add siemens_s7300 subsystem "python /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/ScriptGenEemulate.py /home/jgallenstein/Documents/JustinHoneyd/ScriptGenE/Justin\ pcaps/step7/final_tree_port102 102 -f -i eth0 -e $ipsrc"
set siemens_s7300 ethernet "00:0E:8C:85:AD:0A"
bind 192.168.159.202 siemens_s7300

```

**Figure 25. Honeyd Configuration File**

With the Honeyd honeypots created, the four experiments are conducted to evaluate the research goals.

1. Experiment 1: Utilize Nmap to scan the honeypots and determine accuracy of network layer emulation. Accuracy is determined by comparing each honeypot's Nmap scan to the scan of its respective legitimate PLC.
2. Experiment 2: Determine if Shodan's Honeyscore tool identifies the honeypots as legitimate PLCs.



3. Experiment 3: Using RSLinx and STEP7 software, determine if the honeypots' application-layer can return the same results as the legitimate PLCs.
4. Experiment 4: Use Wget requests to analyze the new best-fit algorithm's ability to return correct results without exact matches.

### 4.3 System Boundaries

As shown in Figure 26, the System Under Test (SUT) is the Honeyd framework. While the main Component Under Test (CUT) is the ScriptGenEemulate.py functionality.

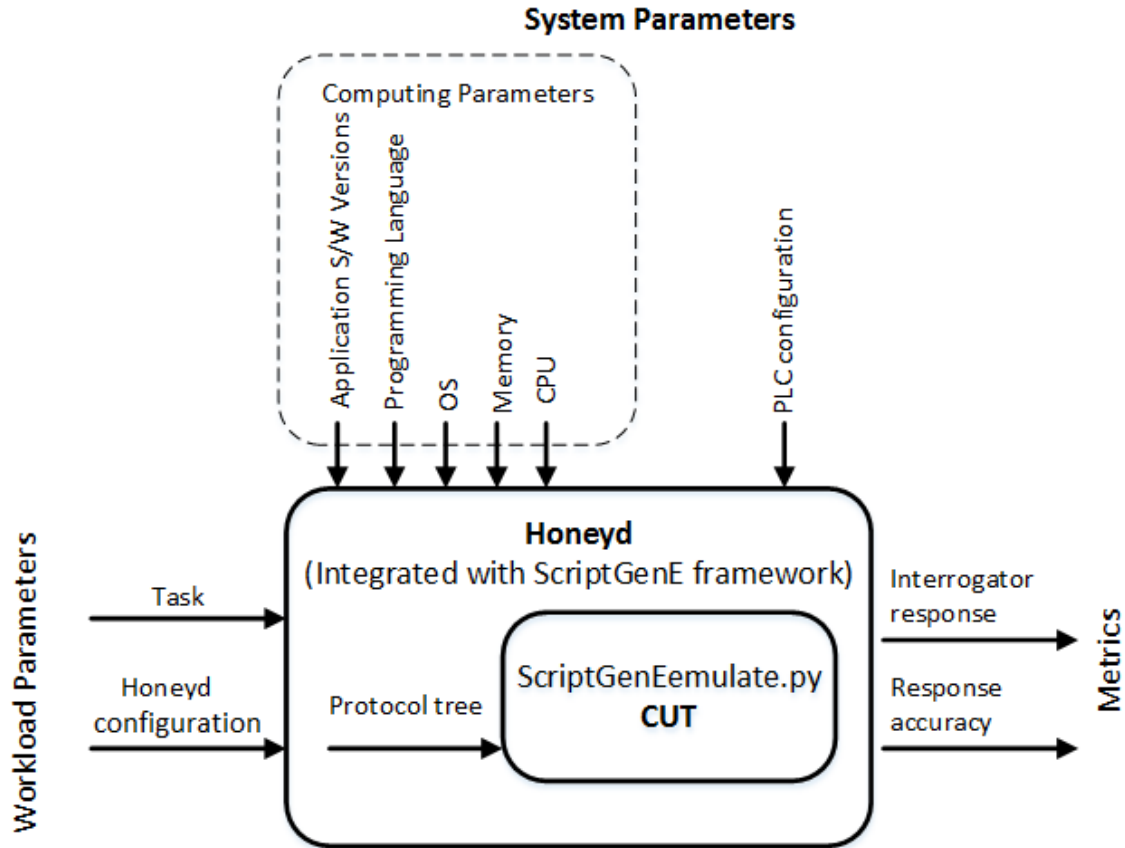


Figure 26. ScriptGenE Emulator Framework

## 4.4 Parameters and Factors

The parameters of this experiment can be divided into two categories, workload and system. Furthermore, parameters that vary are called *factors* with the different variations designated as *levels*. This section describes the applicable parameters and how they impact the SUT.

### 4.4.1 Workload Parameters.

The workload parameters in this experiment can be further divided into two categories —task and configuration. The configuration parameters supply the initialization options to the Honeyd configuration file. This includes a series of p-trees each correlated to the correct port and protocol (e.g., port 44818:ENIP), a designated port configuration, MAC address, and corresponding Nmap personality. The task parameters are used to evaluate emulation competency by utilizing the tools described in Section 2.2.5.

#### 4.4.1.1 Task.

For the first task Nmap is utilized to evaluate the ability of the SUT to emulate the network layer of the target PLC. This test requires scanning both the honeypot and legitimate PLC with the following Nmap command `"nmap ipaddress -T4 -O"`. The `-T4` argument specifies the speed at which to scan the target. This can range from `T1`–`T5`, this value is chosen to ensure that the honeypot can withstand the barrage of traffic typically sent by a scanning tool. The `-O` tells Nmap to attempt to determine the OS details of the device. This is supplied to verify the ability of the Fingerprint.py to create and import a matching OS personality into Honeyd. This test is further extended by utilizing Nmap Scripting Engine (NSE) scripts called `enip-info` and `s7-info`. As shown in Figures 27 and 28, these scripts are written to

determine extra information about their respective PLC type. These scripts gather this information by querying each PLC with protocol (i.e., ISO-TSAP or ENIP) specific commands. The information varies slightly between each script (i.e., enip-info or s7-info) but contains hardware specifications such as serial number, vendor, type, and version.

```
Nmap scan report for 192.168.159.201
Host is up (0.00088s latency).
PORT      STATE SERVICE
44818/tcp  open  EtherNet-IP-2
| enip-info:
|   Vendor: Rockwell Automation/Allen-Bradley (1)
|   Product Name: 1756-EWEB/A
|   Serial Number: 0x9cbe6751
|   Device Type: Communications Adapter (12)
|   Product Code: 125
|   Revision: 4.10
|_  Device IP: 192.168.159.201
MAC Address: 00:1D:9C:BB:C4:DB (Rockwell Automation)
```

Figure 27. Nmap NSE enip-info Results

```
Nmap scan report for 192.168.159.202
Host is up (0.00088s latency).
PORT      STATE SERVICE
102/tcp   open  iso-tsap
| s7-info:
|   Module: 6ES7 315-2EH13-0AB0
|   Basic Hardware: 6ES7 315-2EH13-0AB0
|   Version: 2.5.0
|   System Name: SIMATIC 300 Station
|   Module Type: CPU 315-2 PN/DP
|   Serial Number: S C-V9D757002007
|_  Copyright: Original Siemens Equipment
MAC Address: 00:0E:8C:79:B1:06 (Siemens AG A&D ET)
Service Info: Device: specialized
```

Figure 28. Nmap NSE s7-info Results

Honeyscore is used to evaluate the ability of the fully integrated SUT to conceal its honeypot identity. Honeyscore was designed specifically to ascertain if a device is a honeypot. The IP address of the honeypot is entered into the web-based tool, and the result is displayed in the Hypertext Markup Language (HTML). To complete this test the honeypot needs a public IP address and no firewall in place. To accomplish this the experimental setup is altered by plugging the SUT directly into a cable modem. This avoids any Network Address Translation (NAT) complications that could be encountered by trying to port forward from behind a router.

STEP7 and RSLinx are utilized to verify that Honeyd integration did not interfere with the honeypot's ability to deceive industry standard tools. These tools are supplied the IP address of the applicable honeypot, and will attempt to connect and display the modules of the target PLC. These tools are GUI driven so SikuliX, detailed further in Section 4.6.4, is utilized to automate these tasks.

Finally, the Wget tool is used to test the ability of the SUT to handle unknown client requests. The HTTP is chosen for this experiment because it is an example of a protocol that benefits from non-exact matches, due to the variation in browser user-agent information that allows the server to tailor its response to the client. The task utilizes Wget's ability to set the user-agent to craft requests with varying *levels* of variation from the request stored in the SUT's protocol tree. This task is carried out using the command `"wget -R target.ip.address/index.html --user-agent= variation string"`. The `-R` argument tells Wget to recursively download all linked resources. This allows the PLC's entire web server to be downloaded with one Wget command.

This variation string is randomly selected from a user agent database of 180,000 real world choices [56]. These agents range from the typical browser (e.g., IE and Firefox) to various specific Internet applications (e.g., Lincoln State Web Browser).

The ability to match each request with the right resource determines the ability of the SUT to handle unknown requests.

#### **4.4.1.2 Honeyd Configuration.**

Each honeypot is assigned a different personality appropriate to the PLC it is trying to imitate. This personality includes designating which ports should be opened or closed, MAC and IP address, services, and the OS target. As mentioned in Section 3.4.3, these personalities are created with the Fingerprint.py script that generates a personality based upon a given IP address.

#### **4.4.2 System Parameters.**

As shown in Figure 26 the system parameters consist of both PLC configurations and computing parameters. A detailed description of the experimental setup is in Section 4.7. The host laptop designates the computing parameters. Both the laptop and PLC configurations are:

##### **Hewlett Packard 8570W Workstation**

- Microsoft Windows 7 Service Pack 1
- 2.6GHz Intel Core i7-3720QM processor
- 16GB RAM
- VMware Workstation version 12.1.1 build-3770994

##### **Allen-Bradley ControlLogix1756 (L55) PLC**

- Firmware Version 5.001 Build 1
- Slot 0 - L55 Controller with mode set to REM Run (remote Run)
- Slot 1 - 1756-EWEB EtherNet/IP ENBT

- Slot 2 - 1756-IB16/A DCIN - 16 Point, 24V DC Input
- Slot 3 - 1756-OB8/A DCOUT - 8 Point DC Output
- Slot 4 - 1756-OB8/A DCOUT - 8 Point DC Output

### **Allen-Bradley CompactLogix L23E 1769 PLC**

- Firmware Version 5.001 Build 1
- Slot 0 - L55 Controller with mode set to REM Run (remote Run)
- Slot 1 - Embedded IQ16F - High Speed 24V DC Input
- Slot 2 - Embedded OB16 - 16 Point 24V DC Output
- Slot 3 - Embedded IF4XOF2 - Combo Analog 4 Point Input, 2 Point Output
- Slot 4 - Embedded 2 Channel Quadrature, 4 Channel Single-ended HSC Input

### **Siemens SIMATIC S7-300 PLC**

- Firmware version 2.6
- Slot 2 - CPU 315-2 Controller with one Ethernet port
- Slot 4 - Discrete I/O (DI16xDC24V) - 16 Point 24V DC Input
- Slot 5 - Discrete I/O (DO16xDC24V/0.5A) - 16 Point 24V DC Output
- Slot 6 - Discrete I/O (DO16xAC120V/230V/1A) - 16 Point 120V/230V AC Output
- Slot 7 - Discrete I/O (DO16xRel. AC120V/230V) - 16 Point 120V/230V AC Output
- Slot 8 - Analog I/O (AI8xTC) - 8 Point Thermocouple Analog Input
- Slot 9 - Analog I/O (AI8x16Bit) - 8 Point 16 Bit Analog Input

## 4.5 Performance Metrics

Each of the previously mentioned tasks is evaluated using separate performance metrics, based on the type of results returned by the evaluation tools.

Nmap signature similarity is evaluated by comparing the scan results of the legitimate PLCs to those of the SUT. The tests divide the response into four categories: MAC address, IP address, port specifications, and OS details. These categories are explicitly defined in each Nmap scan summary. Each category is then scored as a PASS or FAIL depending if it exactly matches the legitimate PLC.

The Honeyscore performance is determined by the results of supplying the tool the IP address of the SUT. Honeyscore returns either of the two results shown in Figure 29. Based on this result the test is scored as a PASS or FAIL.

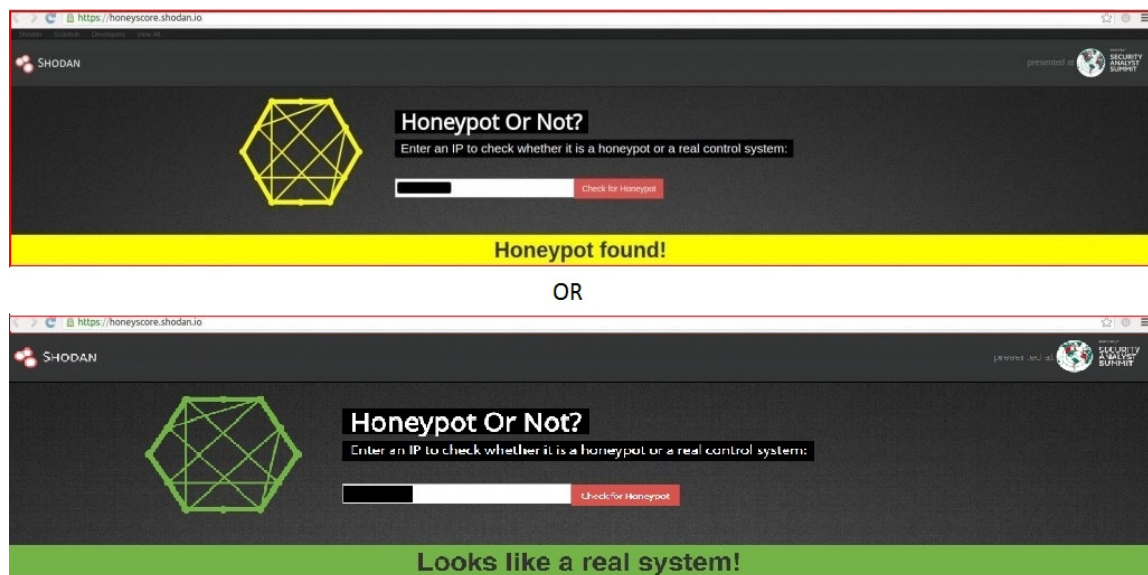


Figure 29. Possible Honeyscore Results

RSLinx and STEP7 performance is evaluated by utilizing SikuliX to view the PLC module hierarchy. As shown in Figure 30, RSLinx has two different possible outcomes. The successful outcome correctly includes all of the appropriate modules for each PLC as described in Section 4.4.2, while the unsuccessful outcome outputs

a red error and simply displays the PLC as an unrecognized device. STEP7 also has two different outcomes when displaying the PLC information. As shown in Figure 31, STEP7 will either successfully display the list of module specifications or will present an error.

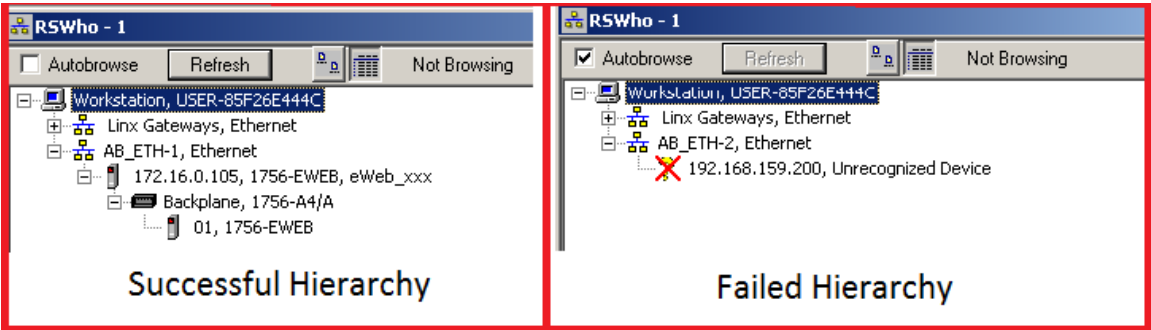


Figure 30. Successful and Failed Module Hierarchy Browsing in RSLinx

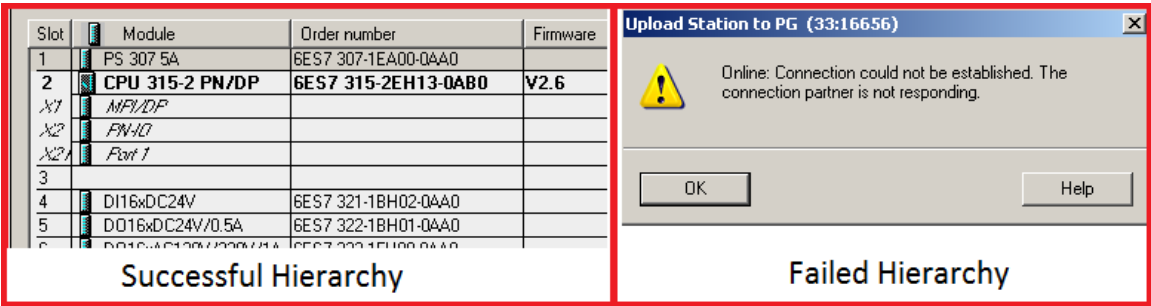


Figure 31. Successful and Failed Module Hierarchy Browsing in STEP7

Determining the results of the Wget experiments is done by evaluating the files downloaded. A request is sent for the desired resource on the legitimate PLC, then that same request is sent to the honeypot. The results are evaluated based on the similarity of the responses. This similarity is determined by running a Linux diff command `diff -s realfile.html emulatedfile.html`, on the files downloaded from both the legitimate and emulated web server. The `-s` option specifies to report when the two files are the same. If the two files match it is designated as a PASS.



## 4.6 Experimental Setup

### 4.6.1 Overview.

As shown in Figure 32, the experimental setup consists of series of virtual machines and PLCs all connected by a switch. Figure 33 shows how this setup is altered to complete the Honeyscore tests as described above in Section 4.4. There are two key differences in this Honeyscore setup. First, the SUT is installed on the bare metal laptop with same software and hardware specifications, instead of within a VM, to avoid VM NAT complications. Second, the SUT is directly connected to a cable modem to provide a non-NAT IP address to Honeyscore.

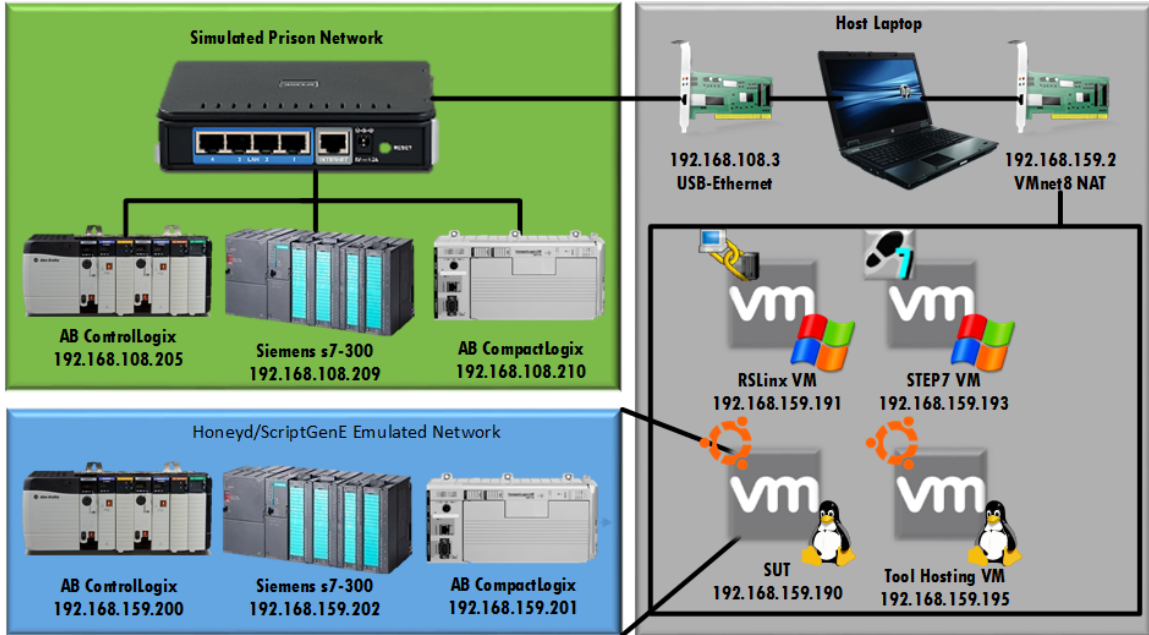


Figure 32. Experiment Setup

### 4.6.2 Machine Configurations.

All physical hardware specifications are detailed in Section 4.4.2. The experiments in total consist of 4 different VMs to handle the various tasks. The primary VM that hosts the SUT is Ubuntu 12.04; the exact specifications are listed in Table 1. The

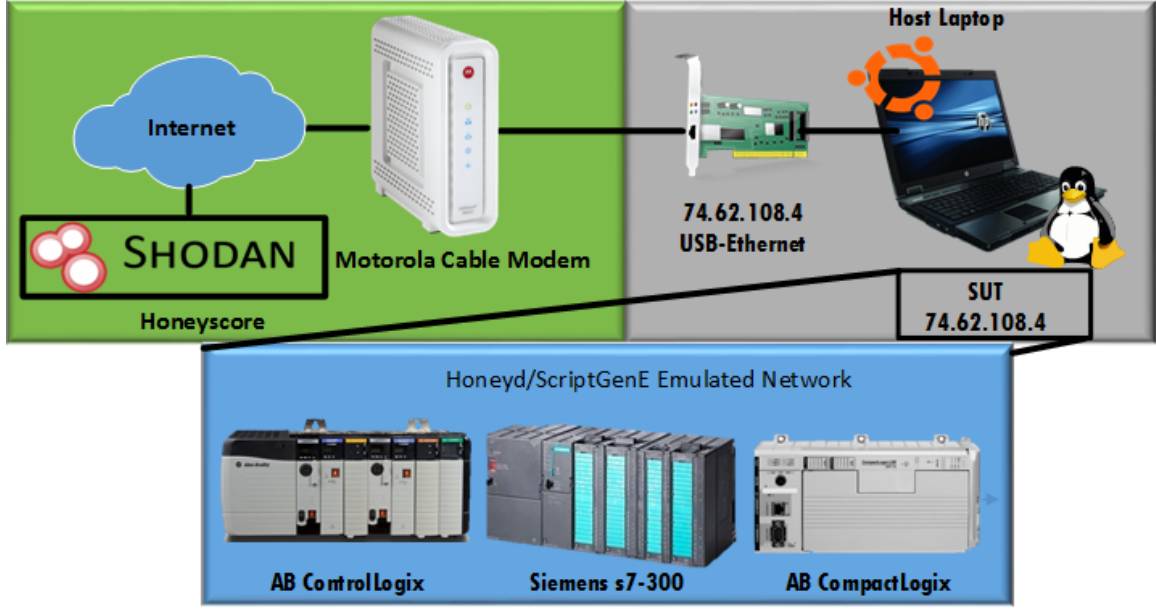


Figure 33. Altered Honeyscore Experiment Setup

version of Honeyd utilized for this experiment is 1.6d. This version of Honeyd was modified by DataSoft from the final official 1.5c Honeyd release to compensate for the changes to Nmap’s OS scanning method. This version also has various custom modifications, as discussed in Section 3.4.5, to allow proper integration with Script-GenE.

Table 1. System Under Test VM Configuration

Ubuntu 12.04			
1 processor core	Linux kernel 3.13.0-32-generic	bash 4.2.25	
2GB RAM	Python 2.7.3	Nmap 7.04	
20GB HD space	gcc 4.6.3	Honeyd 1.6d	

Both Windows XP VMs are nearly identical with the only exception being the software installed. One has the RSLinx software for querying Allen Bradley PLCs (Table 2) while the other has STEP7 software for Siemens PLCs (Table 3).

**Table 2. RSLogix Windows XP VM Configuration**

<b>Windows XP Service Pack 3</b>		
2 processor cores	Python 2.7.2	SikuliX 1.1.0
2GB RAM	Java 1.7.0u25	
60GB HD space	RSLinX Classic 2.59.02	

**Table 3. STEP7 Windows XP VM Configuration**

<b>Windows XP Service Pack 3</b>		
2 processor cores	Python 2.7.2	SikuliX 1.1.0
2GB RAM	Java 1.7.0u25	
60GB HD space	STEP7 5.5	

The last VM is also Ubuntu 12.04 based. This VM acts as a Wget and Nmap interrogator for this experiment. This VM's configuration and software information is listed in Table 4.

**Table 4. Tool Hosting Linux VM Configuration**

<b>Ubuntu 12.04</b>			
2 processor core	Linux kernel 3.13.0-32-generic	wget 1.13.4	
4GB RAM		Nmap 7.04	bash 4.2.25
80GB HD space		Firefox 31.0	

#### 4.6.3 Experimental Scripts.

Experiment automation and data collection is controlled by a custom script for each experiment. These scripts perform the actual experiment and generate files to be analyzed. The scripts were run on the same set of initialized honeypots in the order specified below and include

##### 1. NMAP.py - Ran 30 times

- Runs and saves the respective Nmap scans on all of the PLCs (emulated and legitimate)

- Restarts the Nmap process after each scan

## **2. wget.py - Ran 500 Times**

- Chooses a random user-agent from the database
- Sends the designated Wget request to target server to download files
- Uses a Linux diff command to evaluate each downloaded file as PASS or FAIL. This is done by comparing each file to its target file from the legitimate PLC
- No restart required as each Wget process is killed after downloads complete

## **3. RSLinx.py - Ran 200 times**



- Runs `RsLinxRestart.bat` to restart RSLinx process
- Runs `DeleteDriver.sikuli` to delete previous drivers
- Runs `ConfigDriver.sikuli` to add required driver
- Runs `RSWho.sikuli` to display modules and determine result

## **4. STEP7.py - Ran 30 times**

- Runs `step7restart.bat` to restart STEP7 process
- Runs `STEP7.sikuli` to open STEP7
- Runs `ConfigNode.sikuli` to add the PLC as a node
- Runs `BrowseModules.sikuli` to browse modules
- Runs `DeleteProjectObjects.sikuli` to delete the PLC

#### 4.6.4 GUI Automation.

RSLinx and STEP7 are tools that are operated with a Graphical User Interface (GUI). Therefore, simple command line arguments cannot be used to script interaction. So to automate these experiments the MIT Java project SikuliX was utilized. SikuliX utilizes OpenCV to search the screen for elements and then execute actions [57]. As shown in Figure 34, this is all outlined in a Python script, which consists of code to search for GUI elements on the screen and perform an action (e.g., click, type). If the actions cannot be completed then the script returns a "FAIL".

```
try:
    doubleClick( SIMATIC 300 Station)
    doubleClick()
except FindFailed:
    Debug.user("FAIL - controller or hardware object c
    return "FAIL"


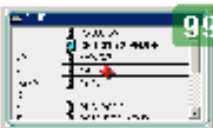
if exists( CPU 315-2 PN/D) or exists(): #
    ... - "FAIL"
```

Figure 34. Sikulix Code Snippet

#### 4.7 Methodology Summary

This chapter details how this research uses different tools to evaluate the integration of Honeyd and ScriptGenE.

## V. Results and Analysis

### 5.1 Overview

The following sections outline the results for each individual task. Overall the tasks demonstrate that the SUT is successfully able to emulate the legitimate PLCs response to the chosen tools.

### 5.2 Metric 1 - Nmap

This experiment was designed to test the network-level emulation of the created honeypots. Each stage of this experiment was carried out 30 times with each honeypot after testing revealed the exact same result each run (i.e., zero confidence interval). The first stage of this experiment involved subjecting all of the emulated PLCs to traditional Nmap scans. As shown in Table 5, the SUT was capable of replicating the identity of the legitimate PLCs. Each column represents a different aspect of the scan including MAC address, IP address, OS type, and port specifications. A PASS in a column represents a successful emulation for all 30 runs. Examples of each PLCs Nmap output compared to its imitation target are shown in Figure 35.

**Table 5. Nmap Experiment Results**

Honeypot	MAC Address	IP Address	OS Type	Ports
192.168.159.200	PASS	PASS	PASS	PASS
192.168.159.201	PASS	PASS	PASS	PASS
192.168.159.202	PASS	PASS	PASS	PASS

(PASS indicates exact emulation match for all 30 runs)

The second stage results in Tables 6 and 7 demonstrate the ability of the SUT to handle the more complex scanning techniques of the NSE scripts. These scripts, as described in Section 4.4.1.1, use protocol specific techniques to provide additional

Emulated	Legitimate
<p>Nmap scan report for 192.168.159.200 Host is up (0.00064s latency). <u>Not shown:</u> 999 filtered ports <u>PORT</u> STATE SERVICE 80/tcp open http 44818/tcp open EtherNetIP-2 MAC Address: 00:1D:9C:79:8C:FC (Rockwell Automation) Warning: OSScan results may be unreliable because we Device type: general purpose Running: Wind River VxWorks OS CPE: cpe:/o:windriver:vxworks OS details: VxWorks</p> <p>Allen-Bradley ControlLogix</p>	<p>Nmap scan report for 192.168.108.205 Host is up (0.0028s latency). <u>Not shown:</u> 999 closed ports <u>PORT</u> STATE SERVICE 80/tcp open http 44818/tcp open EtherNetIP-2 MAC Address: 00:1D:9C:8E:67:51 (Rockwell Automation) Device type: general purpose Running: Wind River VxWorks OS CPE: cpe:/o:windriver:vxworks OS details: VxWorks</p> <p>Allen-Bradley ControlLogix</p>
<p>Nmap scan report for 192.168.159.202 Host is up (0.00s latency). <u>Not shown:</u> 1000 open filtered ports, 999 closed ports <u>PORT</u> STATE SERVICE 102/tcp open iso-tsap MAC Address: 00:0E:8C:F1:45:85 (Siemens AG A&amp;D ET) No exact OS matches for host (If you know what OS is TCP/IP fingerprint:</p> <p>SIEMENS S7-300</p>	<p>Nmap scan report for 192.168.108.209 Host is up (0.000020s latency). <u>Not shown:</u> 1000 open filtered ports, 999 closed ports <u>PORT</u> STATE SERVICE 102/tcp open iso-tsap MAC Address: 00:0E:8C:02:33:82 (Siemens AG A&amp;D ET) No exact OS matches for host (If you know what OS is TCP/IP fingerprint:</p> <p>SIEMENS S7-300</p>
<p>Nmap scan report for 192.168.159.201 Host is up (0.00011s latency). <u>Not shown:</u> 1001 open filtered ports, 1000 filtered port <u>PORT</u> STATE SERVICE 44818/tcp open EtherNetIP-2 MAC Address: 00:00:BC:ED:0B:1F (Rockwell Automation) Warning: OSScan results may be unreliable because we co Device type: general purpose Running: Wind River VxWorks OS CPE: cpe:/o:windriver:vxworks OS details: VxWorks</p> <p>Allen-Bradley CompactLogix</p>	<p>Nmap scan report for 192.168.108.210 Host is up (0.00020s latency). <u>Not shown:</u> 1001 open filtered ports, 1000 filtered por <u>PORT</u> STATE SERVICE 44818/tcp open EtherNetIP-2 MAC Address: 00:00:BC:87:2C:36 (Rockwell Automation) Warning: OSScan results may be unreliable because we Device type: general purpose Running: Wind River VxWorks OS CPE: cpe:/o:windriver:vxworks OS details: VxWorks</p> <p>Allen-Bradley CompactLogix</p>

Figure 35. Basic Nmap Scan Comparisons

information about the target. Once again each column represents a different aspect of the information returned for the emulated PLC. Examples of each PLC’s Nmap NSE output compared to its imitation target are shown in Figure 36. For the IP address, a PASS occurs if the IP address returned by the NSE script matches the target IP address entered into the NMAP scan, a mismatch here would raise suspicion. For all other columns, a PASS is achieved if the response exactly matches the applicable legitimate PLC response.

**Table 6. NSE enip-info Results**

Honeypot	Vendor	Name	Serial	Product #	Revision	Device IP
192.168.159.200	PASS	PASS	PASS	PASS	PASS	PASS
192.168.159.201	PASS	PASS	PASS	PASS	PASS	PASS

(PASS indicates exact emulation match for all 30 runs)

**Table 7. NSE s7-info Results**

Honeypot	Hardware	Name	Version	Type	Module	Serial #
192.168.159.202	PASS	PASS	PASS	PASS	PASS	PASS

(PASS indicates exact emulation match for all 30 runs)

### 5.3 Metric 2 - Honeyscore

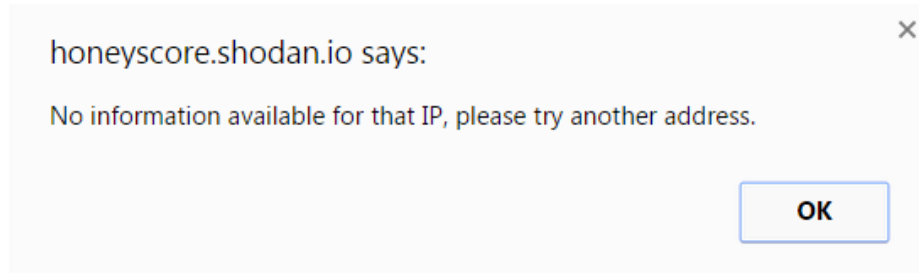
The Honeyscore tool did not perform as expected for this experiment. The SUT was configured to have a direct connection to the Internet. Despite this, Honeyscore was unable to return informative results. Instead, it displayed the message shown in Figure 37. This experiment was carried out 30 times with each emulated target after testing revealed the exact same result every time (i.e., zero confidence interval).

In order to verify that neither the configuration or SUT was at fault, an external connection to the SUT was successfully performed. This connection, made by a mobile device through the cellular network, was able to load the homepage of the PLC. This



Emulated	Legitimate
<p>Nmap scan report for 192.168.159.200 Host is up (0.00088s latency).</p> <p>PORT STATE SERVICE</p> <p>44818/tcp open EtherNet-IP-2</p> <p>  enip-info:</p> <p>  Vendor: Rockwell Automation/Allen-Bradley (1)</p> <p>  Product Name: 1756-ENEB/A</p> <p>  Serial Number: 0x9cbe6751</p> <p>  Device Type: Communications Adapter (12)</p> <p>  Product Code: 125</p> <p>  Revision: 4.10</p> <p>  Device IP: 192.168.159.200</p> <p> _ MAC Address: 00:1D:9C:8B:C4:D8 (Rockwell Automation)</p> <p>Allen-Bradley ControlLogix</p>	<p>Nmap scan report for 192.168.108.205 Host is up (0.00088s latency).</p> <p>PORT STATE SERVICE</p> <p>44818/tcp open EtherNet-IP-2</p> <p>  enip-info:</p> <p>  Vendor: Rockwell Automation/Allen-Bradley (1)</p> <p>  Product Name: 1756-ENEB/A</p> <p>  Serial Number: 0x9cbe6751</p> <p>  Device Type: Communications Adapter (12)</p> <p>  Product Code: 125</p> <p>  Revision: 4.10</p> <p>  Device IP: 192.168.108.205</p> <p> _ MAC Address: 00:1D:9C:8E:67:51 (Rockwell Automation)</p> <p>Allen-Bradley ControlLogix</p>
<p>Nmap scan report for 192.168.159.202 Host is up (0.00088s latency).</p> <p>PORT STATE SERVICE</p> <p>102/tcp open iso-tsap</p> <p>  s7-info:</p> <p>  Module: 6ES7 315-2EH13-0AB0</p> <p>  Basic Hardware: 6ES7 315-2EH13-0AB0</p> <p>  Version: 2.5.0</p> <p>  System Name: SIMATIC 300 Station</p> <p>  Module Type: CPU 315-2 PN/DP</p> <p>  Serial Number: S C-V9D757002007</p> <p> _ Copyright: Original Siemens Equipment</p> <p> _ MAC Address: 00:0E:8C:79:B1:06 (Siemens AG A&amp;D ET)</p> <p> _ Service Info: Device: specialized</p> <p>SIEMENS S7-300</p>	<p>Nmap scan report for 192.168.108.209 Host is up (0.00088s latency).</p> <p>PORT STATE SERVICE</p> <p>102/tcp open iso-tsap</p> <p>  s7-info:</p> <p>  Module: 6ES7 315-2EH13-0AB0</p> <p>  Basic Hardware: 6ES7 315-2EH13-0AB0</p> <p>  Version: 2.5.0</p> <p>  System Name: SIMATIC 300 Station</p> <p>  Module Type: CPU 315-2 PN/DP</p> <p>  Serial Number: S C-V9D757002007</p> <p> _ Copyright: Original Siemens Equipment</p> <p> _ MAC Address: 00:0E:8C:02:33:82 (Siemens AG A&amp;D ET)</p> <p> _ Service Info: Device: specialized</p> <p>SIEMENS S7-300</p>
<p>Nmap scan report for 192.168.159.201 Host is up (0.00s latency).</p> <p>PORT STATE SERVICE</p> <p>44818/tcp open EtherNet-IP-2</p> <p>  enip-info:</p> <p>  Vendor: Rockwell Automation/Allen-Bradley (1)</p> <p>  Product Name: 1769-L23E-QBFC1 Ethernet Port</p> <p>  Serial Number: 0xc01eedb1</p> <p>  Device Type: Communications Adapter (12)</p> <p>  Product Code: 191</p> <p>  Revision: 19.13</p> <p>  Device IP: 192.168.159.201</p> <p> _ MAC Address: 00:00:BC:ED:0B:1F (Rockwell Automation)</p> <p>Allen-Bradley CompactLogix</p>	<p>Nmap scan report for 192.168.159.210 Host is up (0.00s latency).</p> <p>PORT STATE SERVICE</p> <p>44818/tcp open EtherNet-IP-2</p> <p>  enip-info:</p> <p>  Vendor: Rockwell Automation/Allen-Bradley (1)</p> <p>  Product Name: 1769-L23E-QBFC1 Ethernet Port</p> <p>  Serial Number: 0xc01eedb1</p> <p>  Device Type: Communications Adapter (12)</p> <p>  Product Code: 191</p> <p>  Revision: 19.13</p> <p>  Device IP: 192.168.159.210</p> <p> _ MAC Address: 00:00:BC:87:2C:36 (Rockwell Automation)</p> <p>Allen-Bradley CompactLogix</p>

Figure 36. Nmap NSE Script Comparisons



**Figure 37. Honeyscore Results**

demonstrated that the fault lied somewhere outside of this research’s control. Failed attempts were made to contact the tool creator to determine possible reasons for its failure. Further understanding of the functionality of the closed-source Honeyscore tool is required to interpret these results.

#### **5.4 Metric 3 - PLC Module Discovery**

The following results demonstrate the ability of the SUT to handle complex protocol emulation. These tasks are carried out by the appropriate manufacturer’s software (i.e., RSLinx or STEP7). The RSLinx experiment was carried out 200 times for each honeypot, after testing showed a variable success rate. The STEP7 experiment was only carried out 30 times after testing show each result was exactly the same (i.e., zero confidence interval). The results in Figure 38 show the comparisons between the successful emulation and legitimate PLCs. These comparisons show that emulated PLC’s modules, as listed in Section 4.4.2, were correctly displayed to exactly match the modules of their respective legitimate PLC. For RSLinx, the one difference is that the IP address of each emulated PLC correctly matches the IP address assigned in Honeyd, instead of the IP address of its emulation target.

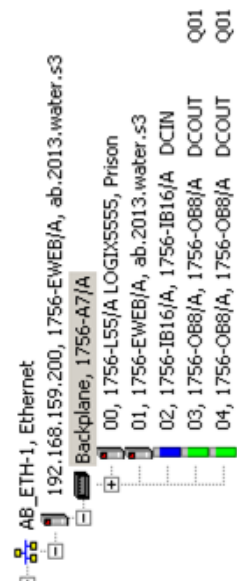
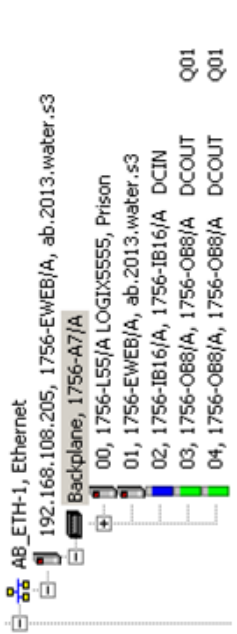
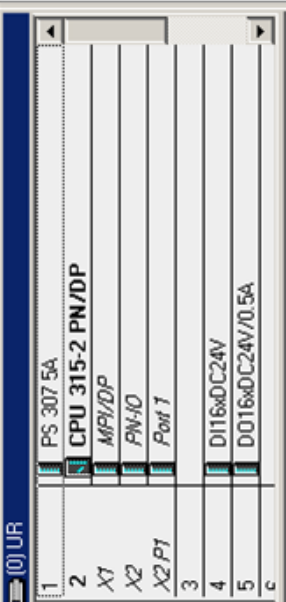
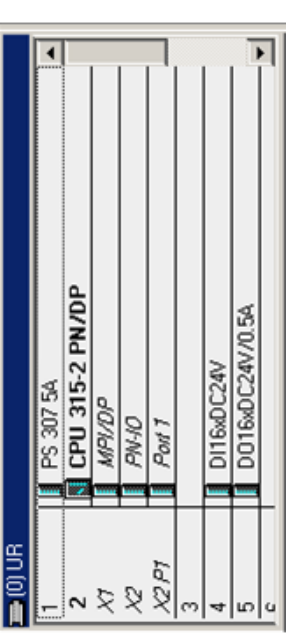
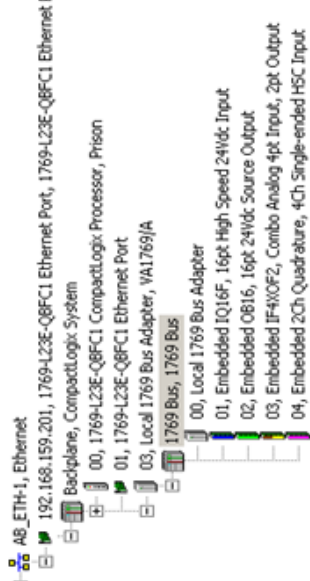
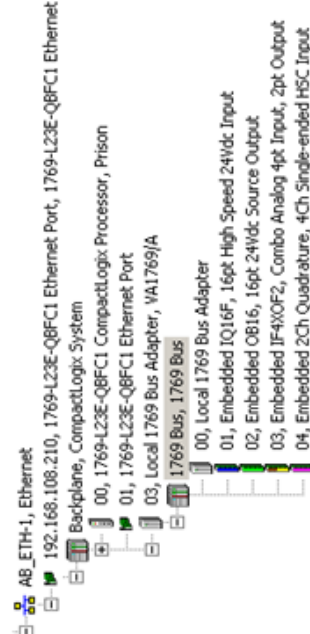
Emulated	Legitimate
 <p>AB_ETH-1, Ethernet 192.168.159.200, 1756-EWEB/A, ab.2013.water.s3 Backplane, 1756-A7/A 00, 1756-L55/A LOGIX5555, Prison 01, 1756-EWEB/A, ab.2013.water.s3 02, 1756-IB16/A, 1756-IB16/A DCIN 03, 1756-OB8/A, 1756-OB8/A DCOU 04, 1756-OB8/A, 1756-OB8/A DCOU Q01 Q01</p> <p>Allen-Bradley ControlLogix</p>	 <p>AB_ETH-1, Ethernet 192.168.108.205, 1756-EWEB/A, ab.2013.water.s3 Backplane, 1756-A7/A 00, 1756-L55/A LOGIX5555, Prison 01, 1756-EWEB/A, ab.2013.water.s3 02, 1756-IB16/A, 1756-IB16/A DCIN 03, 1756-OB8/A, 1756-OB8/A DCOU 04, 1756-OB8/A, 1756-OB8/A DCOU Q01 Q01</p> <p>Allen-Bradley ControlLogix</p>
 <p>PS 307 5A CPU 315-2 PN/DP MPI/DP PN-IO Port 1 DI16xDC24V DO16xDC24V/0.5A</p> <p>SIEMENS S7-300</p>	 <p>PS 307 5A CPU 315-2 PN/DP MPI/DP PN-IO Port 1 DI16xDC24V DO16xDC24V/0.5A</p> <p>SIEMENS S7-300</p>
 <p>AB_ETH-1, Ethernet 192.168.159.201, 1769-L23E-Q8FC1 Ethernet Port, 1769-L23E-Q8FC1 Ethernet Port Backplane, CompactLogix System 00, 1769-L23E-Q8FC1 CompactLogix Processor, Prison 01, 1769-L23E-Q8FC1 Ethernet Port 03, Local 1769 Bus Adapter, VA1769/A 1769 Bus, 1769 Bus 00, Local 1769 Bus Adapter 01, Embedded IQ16F, 16pt High Speed 24Vdc Input 02, Embedded O816, 16pt 24Vdc Source Output 03, Embedded IF4XOF2, Combo Analog 4pt Input, 2pt Output 04, Embedded 2Ch Quadrature, 4Ch Single-ended HSC Input</p> <p>Allen-Bradley CompactLogix</p>	 <p>AB_ETH-1, Ethernet 192.168.108.210, 1769-L23E-Q8FC1 Ethernet Port, 1769-L23E-Q8FC1 Ethernet Port Backplane, CompactLogix System 00, 1769-L23E-Q8FC1 CompactLogix Processor, Prison 01, 1769-L23E-Q8FC1 Ethernet Port 03, Local 1769 Bus Adapter, VA1769/A 1769 Bus, 1769 Bus 00, Local 1769 Bus Adapter 01, Embedded IQ16F, 16pt High Speed 24Vdc Input 02, Embedded O816, 16pt 24Vdc Source Output 03, Embedded IF4XOF2, Combo Analog 4pt Input, 2pt Output 04, Embedded 2Ch Quadrature, 4Ch Single-ended HSC Input</p> <p>Allen-Bradley CompactLogix</p>

Figure 38. Module Display Task Comparison

### 5.4.1 RSLinx Tasks.

The results of the RSLinx tasks for the Allen Bradley PLCs are shown in Table 8. This table is divided into three rows. The first two rows each represent an emulated honeypot, while the last row represents the results from the legitimate PLCs. The .200 honeypot shows a success rate of 82 percent, with an average of 116 seconds to correctly display the PLC and corresponding modules. The .201 exhibits similar results with a success rate of 77 percent and a matching 116 second average time. This is compared to a 100 percent correct display rate with an average time of 10 seconds for the legitimate PLCs.

**Table 8. RSLinx Results**

Honeypot	Avg Success Time (sec)	Success %	Standard Deviation
192.168.159.200	115.965	82%	77.708
192.168.159.201	116.384	77%	81.998
Legitimate	10.023	100%	0.243

The results can be explained by analyzing the p-trees utilized to create the emulated PLCs. These p-trees are built from a PCAP of requests and responses that the RSLinx software sent to the legitimate PLC. As shown in Figure 39, for these legitimate PLCs RSLinx sends only one sequence of the Get Attribute All requests. However, as shown in Figure 40, for the emulated PLCs, RSLinx sometimes sends a series of Send RR Data requests prior to sending the Get Attribute All request that the emulated PLCs expect. For this period of time, the honeypots are unable to correctly respond. As shown in Figure 41, this difference in requests is likely due to a series of resets that the legitimate PLC sends prior to communication. This is different than the honeypots which begin communication with RSLinx's first SYN request.

192.168.159.130	192.168.1.100	ENIP	78 List Services (Req)
192.168.159.130	192.168.1.100	ENIP	78 List Interfaces (Req)
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-2 > 8881 [ACK] Seq=1 Ack=25 win=64240 Len=0
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-2 > 8881 [ACK] Seq=1 Ack=49 win=64240 Len=0
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-2 > 8881 [ACK] Seq=1 Ack=73 win=64240 Len=0
192.168.1.100	192.168.159.130	TCP	120 Ethernet-IP-2 > 8881 [PSH, ACK] Seq=1 Ack=73 win=64240 Len=66
192.168.1.100	192.168.159.130	ENIP	130 List Interfaces (Rsp)
192.168.159.130	192.168.1.100	TCP	54 8881 > Ethernet-IP-2 [ACK] Seq=73 Ack=143 win=64098 Len=0
192.168.159.130	192.168.1.100	ENIP	82 Register Session (Req), Session: 0x00000000
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-2 > 8881 [ACK] Seq=143 Ack=101 win=64240 Len=0
192.168.1.100	192.168.159.130	ENIP	82 Register Session (Rsp), Session: 0x0F020200
192.168.159.130	192.168.1.100	CIP	100 Get Attribute All
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-2 > 8881 [ACK] Seq=171 Ack=147 win=64240 Len=0
192.168.1.100	192.168.159.130	CIP	122 Success
192.168.159.130	192.168.1.100	CIP	100 Get Attribute All
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-2 > 8881 [ACK] Seq=239 Ack=193 win=64240 Len=0
192.168.1.100	192.168.159.130	CIP	145 Success

Figure 39. Attribute All Request Sequence

192.168.159.128	192.168.159.133	ENIP	78 List Services (Req)
192.168.159.128	192.168.159.133	ENIP	78 List Interfaces (Req)
192.168.159.133	192.168.159.128	TCP	60 Ethernet-IP-2 > delta-mcp [ACK] Seq=1 Ack=25 win=29200 Len=0
192.168.159.133	192.168.159.128	TCP	60 Ethernet-IP-2 > delta-mcp [ACK] Seq=1 Ack=49 win=29200 Len=0
192.168.159.133	192.168.159.128	TCP	60 Ethernet-IP-2 > delta-mcp [ACK] Seq=1 Ack=73 win=29200 Len=0
192.168.159.133	192.168.159.128	TCP	120 Ethernet-IP-2 > delta-mcp [PSH, ACK] Seq=1 Ack=73 win=29200 Len=66
192.168.159.128	192.168.159.133	TCP	54 delta-mcp > Ethernet-IP-2 [ACK] Seq=73 Ack=67 win=64174 Len=0
192.168.159.128	192.168.159.255	UDP	82 Source port: afrog Destination port: sentinelsrm
192.168.159.131	255.255.255.255	UDP	82 Source port: blackjack Destination port: sentinelsrm
192.168.159.133	192.168.159.128	ENIP	130 List Interfaces (Rsp)
192.168.159.128	192.168.159.133	ENIP	82 Register Session (Req), Session: 0x00000000
192.168.159.133	192.168.159.128	ENIP	82 Register Session (Rsp), Session: 0x0B020700
192.168.159.128	192.168.159.133	ENIP	115 Send RR Data (Req)

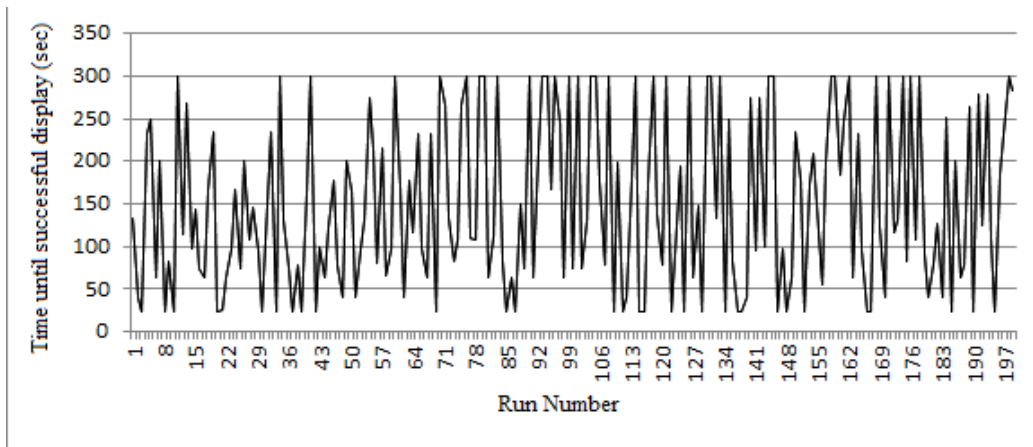
Figure 40. Send RR Data Request Sequence

192.168.159.130	192.168.1.100	TCP	62 cddbp-alt > Ethernet-IP-1 [SYN] Seq=0 win=64240 Len=0 MSS=1460 SACK_PERM=1
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-1 > cddbp-alt [RST, ACK] Seq=1 Ack=1 win=64240 Len=0
192.168.159.130	192.168.1.100	TCP	62 cddbp-alt > Ethernet-IP-1 [SYN] Seq=0 win=64240 Len=0 MSS=1460 SACK_PERM=1
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-1 > cddbp-alt [RST, ACK] Seq=2660556121 Ack=1 win=64240 Len=0
192.168.159.130	192.168.1.100	TCP	62 cddbp-alt > Ethernet-IP-1 [SYN] Seq=0 win=64240 Len=0 MSS=1460 SACK_PERM=1
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-1 > cddbp-alt [RST, ACK] Seq=3094340290 Ack=1 win=64240 Len=0
192.168.159.130	192.168.1.100	TCP	62 8881 > Ethernet-IP-2 [SYN] Seq=0 win=64240 Len=0 MSS=1460 SACK_PERM=1
192.168.1.100	192.168.159.130	TCP	60 Ethernet-IP-2 > 8881 [SYN, ACK] Seq=0 Ack=1 win=64240 Len=0 MSS=1460

Figure 41. Series of Resets by Legitimate PLC

To combat this issue, a simulated RSLinx client was created in Python to query the legitimate PLC with these Send RR Data requests to ascertain the correct response. The hope was that by utilizing these correct responses a better p-tree could be built to handle the protocol. Despite these improved p-trees the emulated PLCs could not respond successfully to the Send RR Data requests from RSLinx. This is likely due to inaccurate emulation of RSLinx by the simulated Python RSLinx client that led to a corrupted p-tree.

To further understand why the variation was so high and to ascertain a pattern in RSLinx requests, this data was graphed as shown in Figure 42. These results show no pattern in the time required to successfully respond. RSLinx was restarted between each run to minimize uncontrolled variables, yet seemingly the series of messages was still random. Further knowledge of how RSLinx determines the sequence of messages to send is required to enhance this protocols emulation.



**Figure 42. RSLinx Module Display Graph for 192.168.159.200**

#### **5.4.2 STEP7 Tasks.**

The results for the STEP7 software are much more consistent than those for RSLinx. As shown in Table 9, the emulated Siemens honeypots were able to correctly display with 100 percent accuracy. There is no noticeable difference between time

recorded for the honeypot versus the legitimate PLC, as both are within each other’s standard deviation.

**Table 9. STEP7 Results**

Honeypot	Avg Success Time (sec)	Success %	Standard Deviation
192.168.159.202	4.2	100%	0.32
Legitimate	4.065	100%	0.25

These results further show the importance of an accurate p-tree. STEP7 uses a more advanced protocol for communication with the PLC. However, the honeypot was significantly more successful compared to the Allen-Bradley PLCs. This is because although the protocol is more advanced, it is also more consistent. When the PCAPs of the experiment are investigated, it is seen that STEP7 organizes its discovery and connection requests in the same order every connection. This consistency allows an accurate p-tree to be created to emulate the protocol. With an accurate p-tree the honeypot can emulate this complex ISO-TSAP protocol even better than the simpler ENIP protocol.

## 5.5 Metric 4 - Best-Fit Algorithm

This experiment evaluates the new best-fit algorithm compared to the previous iterations matching algorithm. As described in Section 4.2, this test involves querying the emulated PLC’s web server with GET requests, each containing a different user-agent from a database of real world examples. This experiment was carried out 500 times, in order to ensure a wide variety of random user-agent selections. The selection database had some slight variations of the same particular user-agent. By selecting 500 user-agents it was ensured that the sample size would consist of sufficient variation. The total file downloads for each ScriptGenE version is 29,500, because

**Table 10. Wget Results**

ScriptGenE version	Wget Requests	Correct Downloads	Files Downloaded	Success %
New	500	20001	29500	67.8%
Previous	500	0	29500	0%
New Non-Integrated	500	29500	29500	100%

each Wget request sends an additional 59 requests for files, as it recursively downloads linked resources.

As Table 10 shows, the new algorithm integrated with Honeyd was able to match unknown requests with 67.8 percent accuracy. While the previous matching algorithm was unable to match any of the requests with their correct resource. This is expected because the previous algorithm only finds exact matches to the requests stored in the p-tree. This table also shows a success rate of 100 percent when the ScriptGenE framework was tested alone (i.e., not integrated with Honeyd). This test was carried out after noticing that all of the failures occurred when Honeyd did not transfer the received data to the ScriptGenE subsystem. This failed transfer required the algorithm to find an approximate match based on no received data, which resulted in a failure.

After observing the standalone ScriptGenE is able to match all requests successfully, further inspection was required. It was discovered that Honeyd was unable to keep up with the sheer amount of connections. This can be seen clearly by inspecting the results of loading the emulated web server in a browser. As shown in Figure 43, the new integrated system failed to resolve some of the resources on the page (e.g., header logos and sidebar CSS formatting), while the standalone system performed flawlessly. This test demonstrates the overhead required for Honeyd to retrieve network traffic, choose designated subsystem, transfer the data, and run the ScriptGenEemulate Python script. This overhead is encountered on every connection, and this PLC homepage requires 59 simultaneous connections. If one of these data



transfers is missed, then the data is lost. This is, as shown in Figure 44, Honeyd immediately sends an acknowledgment of a packet to the sender, before guaranteeing that the subsystem received the data. This means that there is no attempt to resend.

The links can be traversed on the new integrated homepage to further demonstrate that the problem is with the connection overload and not emulation. When reloaded these links correctly direct the client to the right resources; this is because at that time the influx of connections has ceased.

Another smaller contributing factor to the integrated system's failure to display the PLC homepage is the stateless nature of the HTTP protocol. As shown in Figure 45, the p-trees for HTTP and ISO-TSAP differ significantly in structure. With ISO-TSAP, each node has specific edges to consider for the next matching attempt. With HTTP, all root edges must be considered, thus resulting in longer computation cycles. If these cycles are being processed when a new connection is received possible data loss can occur.



Figure 43. Web Server Result Comparison

29	5.172793	192.168.159.1	192.168.159.200	HTTP	415 GET /css/navtree.css HTTP/1.1
30	5.173652	192.168.159.1	192.168.159.200	HTTP	403 GET /scripts/navtree.js HTTP/1.1
31	5.173809	192.168.159.1	192.168.159.200	HTTP	426 GET /images/paper.gif HTTP/1.1
32	5.176175	192.168.159.200	192.168.159.1	TCP	66 80 → 10931 [ACK] Seq=1 Ack=362 Win=64000 Len=0 TSval=0 TSecr=0
33	5.176176	192.168.159.200	192.168.159.1	TCP	66 80 → 10932 [ACK] Seq=1 Ack=350 Win=64000 Len=0 TSval=0 TSecr=0
34	5.176176	192.168.159.200	192.168.159.1	TCP	66 80 → 10933 [ACK] Seq=1 Ack=373 Win=64000 Len=0 TSval=0 TSecr=0
35	5.204367	192.168.159.1	192.168.159.200	HTTP	418 GET /images/menueendon.gif HTTP/1.1
36	5.204670	192.168.159.1	192.168.159.200	HTTP	415 GET /images/border.gif HTTP/1.1
37	5.220361	192.168.159.200	192.168.159.1	TCP	66 80 → 10934 [ACK] Seq=1 Ack=365 Win=64000 Len=0 TSval=0 TSecr=0
38	5.220362	192.168.159.200	192.168.159.1	TCP	66 80 → 10935 [ACK] Seq=1 Ack=362 Win=64000 Len=0 TSval=0 TSecr=0

Figure 44. Honeyd Data Acknowledgment

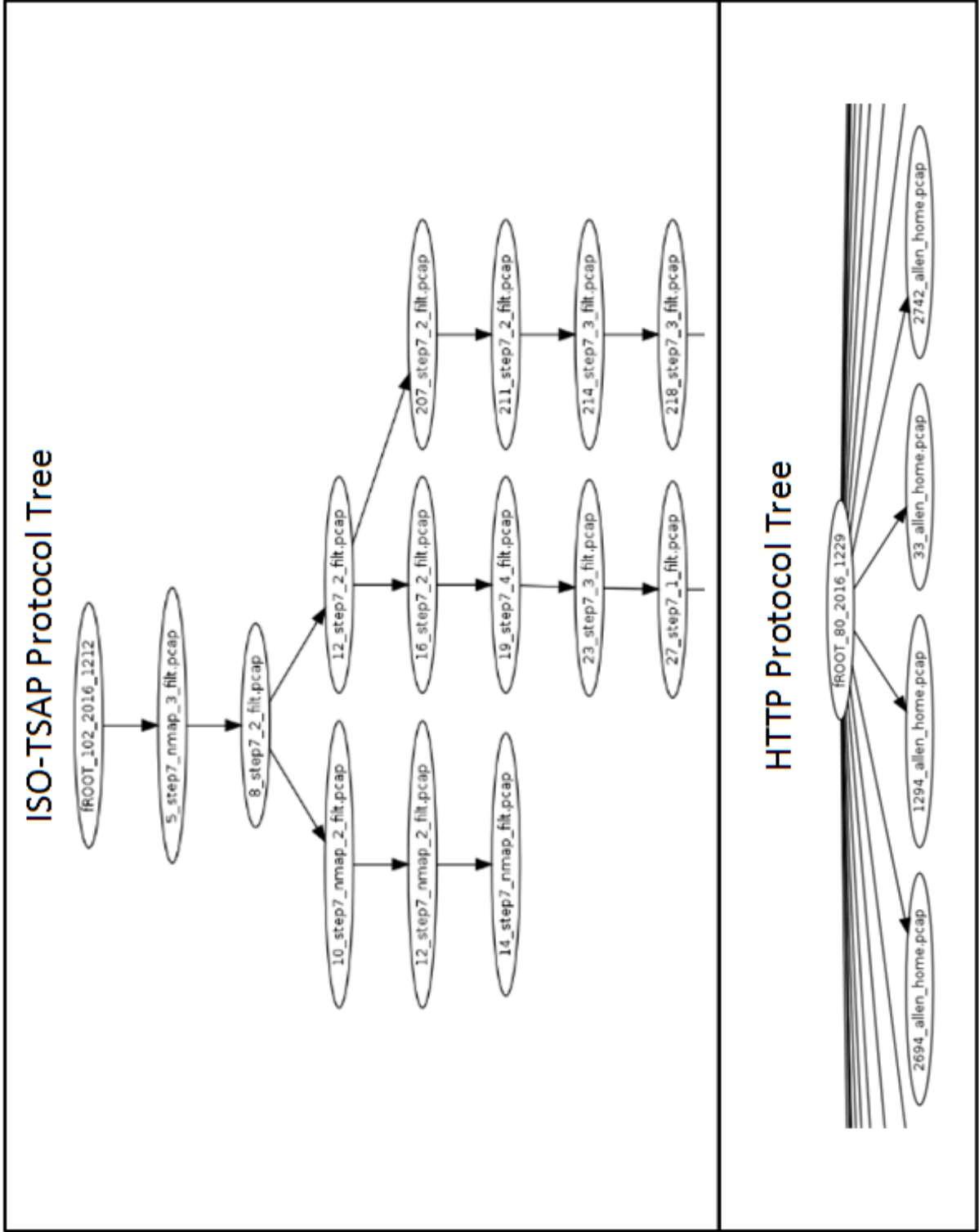


Figure 45. Protocol Tree Comparison

## VI. Conclusions

### 6.1 Introduction

This chapter presents a summary of the research conclusions, impact, and future work. Section 6.2 gives conclusions from the experiment results. Section 6.3 discusses the impact of the research. Section 6.4 presents several ideas and recommendations for future work and related research.

### 6.2 Research Conclusions

This research successfully provides an enhanced PLC emulator based on the application-layer replay framework found in ScriptGenE. The outcome of each research goal from Section 4.1 is discussed below. Each goal is satisfied successfully as shown through the conducted experiments.

#### 6.2.1 Integrated Emulation Performance.

Girtz's research showed a success rate of 33 percent and 0 percent for non-proxied ENIP and ISO-TSAP traffic [58]. This research's success rates of 77 percent and 100 percent for the ENIP and ISO-TSAP application layer protocols increased this by 34 and 100 percent respectively. The emulated honeypots were also able to correctly emulate the network layer of the legitimate PLCs with 100 percent accuracy. These results combined to allow successful emulation of the training prison's industrial network, without relying on a back-end proxy PLC.

#### 6.2.2 Best-Fit Algorithm.

The new best-fit algorithm was shown to correctly select the correct response with 100 percent accuracy for the HTTP protocol. This is improved over the 0 percent non-

proxied accuracy for previous research. One example of this functionality in practice, is successful emulation of a web server regardless of the attacker's browser. This algorithm was developed to be protocol agnostic so it should also improve emulation of other protocols.

### **6.3 Significance of Research**

This research was carried out to further develop a tool that can be used to protect the nation's critical infrastructure.

#### **6.3.1 Contributions.**

The primary contributions of this research are the integration of Honeyd with the ScriptGenE framework, improvements to ScriptGenE functionality, and automatic honeypot configuration.

The most significant overall enhancement is the integration of the Honeyd and ScriptGenE frameworks, which now provides both a network and application layer honeypot capable of emulating a legitimate PLC. This framework includes the `Fingerprint.py` script to automatically configure the honeypot instances.

The main contribution to the ScriptGenE framework is the improvement of the matching algorithm. Previously unknown traffic was unable to be matched, even with just small variations. With the new improvements, testing showed unknown HTTP messages can be matched with 100 percent accuracy.

#### **6.3.2 Applications.**

Now that the two frameworks have been integrated, the SUT can easily be used to provide a cheap, accurate, and simple way to emulate expensive PLC devices. This can be utilized in various industries including industry and academia.

## 6.4 Future Work

The research described in this thesis has significantly improved a framework designed to automatically emulate PLCs. Despite these improvements, future work can be done to further develop it into a more reliable and consistent product.

### 6.4.1 Testing.

Further testing of ScriptGenEemulate.py is required to show it can be deployed in a production environment. Testing with additional protocols, tasks, and PLC configurations can show the emulator is able to handle diverse network conditions.

### 6.4.2 Enhancing ScriptGenEemulate.py Algorithms.

The following sections describe the variety of ways the current ScriptGenEemulate implementation can be enhanced.

#### 6.4.2.1 UDP Support.

Currently UDP support is in place but with trivial testing or hardening. Further evaluation is required to evaluate effectiveness. Besides evaluation, there are multiple areas to improve upon the UDP functionality. One advancement is to include the ability to handle out of order UDP traffic by linking request/response packets based on data payload. Currently the PCAP must be organized so that each UDP request packet corresponds to the next response packet. Another key improvement in this area is to fix the bugs in Honeyd so that ScriptGenE UDP support can operate within an emulated honeypot. Currently Honeyd does not correctly transfer data to the UDP subsystem. Similar changes to those made by this research to the TCP support will likely need to be made.

#### **6.4.2.2 Unknown Transition Algorithm.**

Future research can be done to enhance the emulators ability to quickly select a best match, when an exact match is not available. The speed of this algorithm can likely be improved with various optimizations. Also further testing with other protocols is required to verify the protocol agnostic nature of this algorithm.

#### **6.4.3 Honeyd Connection Overload.**

Experimentation showed that high connection quantities can cause Honeyd to fail to transfer the received network traffic to the designated subsystem. This significantly hampered the ability of the integrated SUT to achieve high levels of accuracy. Two possible solutions are suggested by this research. First, the Honeyd logic could be altered to hold the acknowledgment of a packet until it is verified that the subsystem received it. This will allow the client to resend lost data so the subsystem can eventually find a match. Alternatively, Honeyd could be altered to have a timeout period where it solely focuses on handling connections. During this time all resources are spent receiving data from connections. This could protect against data loss due to a large influx of connections. Future research should investigate and fix this issue to greatly enhance the credibility of this framework.

#### **6.4.4 Manual P-tree Editing.**

This research further enhanced the SUT's ability to automatically create a PLC honeypot. However, in some instances human intuition can outperform this automation. The ability to manually edit a created p-tree with the correct information would be valuable to the emulation process.



## 6.5 Chapter Summary

This chapter presents the conclusions and impact of the ScriptGenEmulate research. To conclude, several recommendations for future work are provided which build upon and extend the emulation techniques developed in this research.

## Bibliography

1. Keith A. Stouffer, Joseph A. Falco, and Karen A. Scarfone. Guide to Industrial Control Systems (ICS) security: Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS), and other control system configurations such as Programmable Logic Controllers (PLC). Technical Report SP 800-82, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011. Retrieved 23 June, 2016 from <http://csrc.nist.gov/publications/nistpubs/800-82/SP800-82-final.pdf>.
2. Nicolas Falliere. Exploring Stuxnet’s PLC infection process. Technical report, Symantec Official Blog, 2010. Retrieved 23 June 2016 from <http://www.symantec.com/connect/blogs/exploring-stuxnet-s-plc-infection-process>.
3. Yu-Lun Huang, Alvaro A. Cárdenas, Saurabh Amin, Zong-Syun Lin, Hsin-Yi Tsai, and Shankar Sastry. Understanding the physical and economic consequences of attacks on control systems. *International Journal of Critical Infrastructure Protection*, 2(3):73–83, 2009.
4. Niels Provos and Thorsten Holz. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Pearson Education, Boston, MA, USA, first edition, 2007.
5. Niels Provos. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, 2004.
6. Miles A. McQueen and Wayne F. Boyer. Deception used for cyber defense of control systems. In *Human System Interactions, 2nd Conference on*, HSI’09, pages 624–631. IEEE, May 2009.
7. Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation. In *Building Analysis Datasets and Gathering Experience Returns for Security, Proceedings of the First Workshop on*, BADGERS ’11, pages 29–36, New York, NY, USA, 2011. ACM.
8. Niels Provos. Honeyd (version 1.6d), 2013. Retrieved 23 August 2016 from <https://github.com/DataSoft/Honeyd>.
9. Xiangdong Li and Li Chen. A survey on methods of automatic protocol reverse engineering. In *Computational Intelligence and Security, Seventh International Conference on*, CIS’11, pages 685–689. IEEE, 2011.
10. Corrado Leita, Ken Mermoud, and Marc Dacier. ScriptGen: an automated script generation tool for Honeyd. In *Computer Security Applications Conference, 21st Annual*, pages 214–223. IEEE, December 2005.

11. Phillip C. Warner. Automatic configuration of programmable logic controller emulators. Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, USA, March 2015. [oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA620212](http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA620212).
12. Michael Winn, Mason Rice, Juan Lopez, and Barry Mullins. Constructing cost-effective and targetable industrial control system honeypots for production networks. *International Journal of Critical Infrastructure Protection*, 10:47–58, May 2015.
13. Weidong Cui, Vern Paxson, and Nicholas Weaver. GQ: Realizing a system to catch worms in a quarter million places. Technical Report 06-004, International Computer Science Institute, September 2006. Retrieved 31 August 2016 from <http://www.icir.org/vern/papers/gq-techreport.pdf>.
14. Corrado Leita and Marc Dacier. SGNET: A worldwide deployable framework to support the analysis of malware threat models. In *Dependable Computing Conference, Seventh European*, pages 99–109. IEEE, May 2008.
15. Walt Boyes, David Clayton, and Inderpreet Shoker. Top 50 automation companies of 2011, 2012. Retrieved 16 October 2016 from <http://www.controlglobal.com/articles/2012/boyes-clayton-serene-economic-recovery/?start=1>.
16. One Hundred Seventh Congress of the United States of America. Section 1016 of the United States Patriot Act of 2001, 2001.
17. White House Office of the Press Secretary. Presidential Policy Directive 21. *Critical Infrastructure Security and Resilience*, 2013. Retrieved 23 June 2016 from <http://www.whitehouse.gov/the-press-office/2013/02/12/presidential-policy-directive-critical-infrastructure-security-and-resil>.
18. John Matherly. Shodan, 2016. Retrieved 13 August 2016 from <http://shodan.io/>.
19. Éireann P. Leverett. Quantitatively assessing and visualising industrial system attack surfaces, June 2011. Retrieved 8 February 2016 from <http://www.cl.cam.ac.uk/~fms27/papers/2011-Leverett-industrial.pdf>.
20. Kelly J. Higgins. ‘Project SHINE illuminates sad state of SCADA/ICS security on the net. Dark Reading blog entry, 2013. Retrieved 23 June 2016 from <http://www.darkreading.com/vulnerabilities---threats/project-shine-illuminates-sad-state-of-scada-ics-security-on-the-net/d/d-id/1140691>.
21. Paul M. Williams. Distinguishing Internet-facing ICS devices using PLC programming information. Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, USA, June 2014 (ADA602989). Retrieved 23 June, 2016 from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA602989>.

22. Kyle Wilhoit. Who's really attacking your ICS equipment? Technical report, Trend Micro, Inc., 2013. Retrieved 23 June 2016 from <http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-whos-really-attacking-your-ics-equipment.pdf>.
23. Kyle Wilhoit. The SCADA that didn't cry wolf. Technical report, Trend Micro, Inc., 2013. Retrieved 23 June 2016 from <http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-the-scada-that-didnt-cry-wolf.pdf>.
24. Roland C. Bodenheimer. Impact of the Shodan computer search engine on Internet-facing Industrial Control System devices. Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, USA, March 2014 (ADA601219). Retrieved 23 June, 2016 from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA601219>.
25. Ruben Santamarta. Project Basecamp – attacking ControlLogix. In *5th SCADA Security Scientific Symposium*, Miami Beach, FL, USA, January 2012. Digital Bond. Retrieved 23 June 2016 from [http://reversemode.com/downloads/logix-report\\_basecamp.pdf](http://reversemode.com/downloads/logix-report_basecamp.pdf).
26. David P. Duggan, Michael Berg, John Dillinger, and Jason Stamp. Penetration testing of Industrial Control Systems. Technical Report 2005-2846P, Sandia National Laboratories, March 2005. Retrieved 26 October 2016 from [http://energy.sandia.gov/wp/wp-content/gallery/uploads/sand\\_2005\\_2846p.pdf](http://energy.sandia.gov/wp/wp-content/gallery/uploads/sand_2005_2846p.pdf).
27. Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, 1999. Retrieved 23 June 2016 from <http://tools.ietf.org/html/rfc2616>.
28. Jon Postel. RFC 791: Darpa internet program protocol specification, 1981. Retrieved 23 June, 2016 from <http://www.ietf.org/rfc/rfc0791.txt>.
29. Fred Cohen. The deception toolkit home page and mailing list, 2012. Retrieved 22 Jan 2017 from <http://all.net/dtk/>.
30. Tom Liston. Labrea: "sticky" honeypot and ids, 2003. Retrieved 22 Jan 2017 from <http://labrea.sourceforge.net/labrea-info.html>.
31. George Bakos. Tiny honeypot, 2013. Retrieved 22 Jan 2017 from <https://github.com/tiny-honeypot/thp>.
32. Ryan McGeehan. The "google hack" honeypot. Retrieved 22 Jan 2017 from <http://ghh.sourceforge.net/>, year = 2007.
33. Laurent Oudot. Php honeypot project. Retrieved 22 Jan 2017 from <http://rstack.org/phphop>, year = 2006.

34. Gordon Lyon. Nmap, 2016. Retrieved 18 August 2016 from <https://nmap.org/book/man-host-discovery.html>.
35. John Matherly. Honeyscore, 2016. Retrieved 20 August 2016 from [honeyscore.shodan.io](https://honeyscore.shodan.io).
36. GNU Project. Wget, 2016. Retrieved 20 August 2016 from <https://www.gnu.org/software/wget/>.
37. Bradley Allen. Rslinx classic, 2016. Retrieved 20 August 2016 from [http://literature.rockwellautomation.com/idc/groups/literature/documents/gr/linux-gr001\\_-en-e.pdf](http://literature.rockwellautomation.com/idc/groups/literature/documents/gr/linux-gr001_-en-e.pdf).
38. Siemens. Step7, 2016. Retrieved 21 August 2016 from <http://w3.siemens.com/mcms/simatic-controller-software/en/step7/Pages/Default.aspx>.
39. Xuxian Jiang and Dongyan Xu. BAIT-TRAP: a catering honeypot framework. Technical report, Purdue University, 2004. Retrieved 23 August, 2016 from <http://friends.cs.purdue.edu/pubs/BaitTrap.pdf>.
40. Vishal Chowdhary, Alok Tongaonkar, and Tzi-cker Chiueh. Towards automatic learning of valid services for honeypots. In *Distributed Computing and Internet Technology, Proceedings of the First International Conference on*, ICDCIT'04, pages 469–469, Berlin, Heidelberg, 2004. Springer-Verlag. Retrieved 31 August, 2016 from <http://seclab.cs.sunysb.edu/alok/papers/icdcit04.pdf>.
41. Deanna R. Fink. Toward automating web protocol configuration for a programmable logic controller emulator. Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, USA, June 2014. Retrieved 24 August 2016 from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA602990>.
42. Weidong Cui, Vern Paxson, Nicholas Weaver, and Randy H. Katz. Protocol-independent adaptive replay of application dialog. In *Network and Distributed System Security Symposium*, February 2006. Retrieved 1 September 2016 from <http://www.internetsociety.org/doc/protocol-independent-adaptive-replay-application-dialog>.
43. Corrado Leita, Marc Dacier, and Frederic Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots. In *Recent Advances in Intrusion Detection, Proceedings of the 9th International Conference on*, RAID'06, pages 185–205, Berlin, Heidelberg, 2006. Springer-Verlag.
44. Christian Kreibich, Nicholas Weaver, Chris Kanich, Weidong Cui, and Vern Paxson. GQ: Practical containment for measuring modern malware systems. In *Internet Measurement Conference, Proceedings of the 2011 ACM SIGCOMM Conference on*, IMC '11, pages 397–412, New York, NY, USA, 2011. ACM.

45. Mariano Graziano, Corrado Leita, and Davide Balzarotti. Towards network containment in malware analysis systems. In *Computer Security Applications Conference, Proceedings of the 28th Annual*, pages 339–348. ACM, 2012.
46. Christian Martin Fuchs and Martin Brunner. Towards next generation malware collection and analysis. *Advances in Security, International Journal on*, 6(1 and 2):32–48, 2013. Retrieved 31 August 2016 from [http://www.thinkmind.org/download.php?articleid=sec\\_v6\\_n12\\_2013\\_3](http://www.thinkmind.org/download.php?articleid=sec_v6_n12_2013_3).
47. Lukas Rist, Johnny Vestergaard, Daniel Haslinger, and Andrea Pasquale. Conpot, 2013. Retrieved 27 August 2016 from <http://conpot.org/>.
48. Digital Bond. SCADA honeynet, 2006. Retrieved 27 August 2016 from <http://www.digitalbond.com/tools/scada-honeynet/>.
49. Daniel Buza, Ferenc Juhasz, Gyorgy Miru, Mark Felegyhazi, and Tamas Holczer. CryPLH: Protecting smart energy systems from targeted attacks with a PLC honeypot. In *Proceedings of Smart Grid Security*, pages 181–192, February 2014. Retrieved 23 June 2016 from <http://crysys.hu/publications/files/BuzaJMFH2014smartgridsec.pdf>.
50. Robert M. Jaromin. Emulation of industrial control field device protocols. Master’s thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, USA, March 2013. Retrieved 24 August 2016 from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA582482>.
51. Python Software Foundation. Python object serialization, 2016. Retrieved 29 August 2016 from <https://docs.python.org/2/library/pickle.html>.
52. Marshall Beddoe. The protocol informatics project, 2004. Retrieved 7 October, 2016 from <http://www.4tphi.net/~awalters/PI/PI.html>.
53. W John Ratcliff and E David Metzener. Pattern matching: The gestalt approach, 1988. Retrieved 31 August 2016 from <http://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/1988/8807/8807c/8807c.htm>.
54. Datasoft. Open source nova version, 2015. Retrieved 29 August 2016 from <https://github.com/DataSoft/Nova>.
55. Joseph Daoud. Multi-plc exercise environments for training ics first responders. Master’s thesis, Air Force Institute of Technology, Wright-Patterson AFB OH, USA, March 2017.
56. Browser Capabilities Project. browscap.csv, 2016. Retrieved 29 August 2016 from <http://browscap.org/>.
57. Raimund Hocke. SikuliX powered by RaiMan, 2016. Retrieved 9 March 2016 from <http://www.sikulix.com>.

58. Kyle Girtz, Mason Rice, Juan Lopez, and Barry Mullins. Practical application layer emulation in industrial control system honeypots. *Critical Infrastructure Protection X*, 2:83–98, March 2016.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. <b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b></p>						
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE		3. DATES COVERED (From — To)		
23-03-2017		Master's Thesis		Sept 2015 — Mar 2017		
4. TITLE AND SUBTITLE  Integration of the Network and Application Layers of Automatically Configured Programmable Logic Controller Honeypots				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)  Justin K. Gallenstein, 2d Lt, USAF				5d. PROJECT NUMBER  17G310		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER  AFIT-ENG-MS-17-M-029		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)  Department of Homeland Security ICS-CERT POC: Neil Hershfield, DHS ICS-CERT Technical Lead ATTN: NPPD/CS&C/NCSD/US-CERT Mailstop: 0635, 245 Murray Lane, SW, Bldg 410, Washington, DC 20528 Email: ics-cert@dhs.gov phone: 1-877-776-7585				10. SPONSOR/MONITOR'S ACRONYM(S)  DHS ICS CERT		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT  DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES  This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.						
14. ABSTRACT This research further develops ScriptGenE, a protocol-agnostic framework capable of accurately creating PLC honeypots. ScriptGenE uses previously captured PLC traffic to create a tree of the protocol and selectively respond to application layer requests in an accurate way. This research integrates ScriptGenE with Honeyd to provide the PLC honeypots with an accurate network layer. This combination provides a comprehensive PLC honeypot. Testing is done by using the combined framework to emulate a network of Allen-Bradley ControlLogix, Allen-Bradley CompactLogix, and Siemens S7-300 PLCs. A series of tools are used to evaluate the legitimacy of the emulated PLC network including Nmap, Honeyscore, RSLinx, STEP7, and Wget. Nmap and Honeyscore are used to show that the combined framework is able to accurately emulate the network layer of three different PLC types with 100 percent accuracy. Using Wget, RSLinx, and STEP7 this research shows the ability to emulate more advanced application layer protocols such as ENIP, ISOTASP, and HTTP with accuracies of 78, 100, and 67 percent respectively.						
15. SUBJECT TERMS  SCADA, honeypot, programmable logic controller, industrial control systems, automation, emulator, protocol reverse engineering						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Barry E. Mullins (ENG)	
U	U	U	U	104	19b. TELEPHONE NUMBER (include area code) (937) 255-3636 x7979 Barry.Mullins@afit.edu	